Intégration et évaluation de comportements d'agents dans les jeux vidéo

---

Rapport de stage de master recherche en informatique

Fabien Tencé (fabien.tence@ens.insa-rennes.fr)

Encadrant :
Cédric Buche (buche@enib.fr)

Laboratoire :
Centre Européen de Réalité Virtuelle (CERV) équipe ARéVi
Laboratoire d'Informatique des Systèmes Complexes
(LISyC, EA 3883)

6 juin 2008

Table des matières

1 Introduction **1**

2 Intégration de comportements d'agents aux jeux vidéo **2**
2.1 Présentation des problématiques et état de l'art 2
2.1.1 Gestion des échanges entre le jeu et l'agent 2
2.1.2 Approches client et serveur 3
2.1.3 Solutions existantes pour l'intégration 5
2.1.4 Pogamut 2 7
2.1.5 Un agent dans Pogamut 2 11
2.2 Étude et travaux sur l'interopérabilité de Pogamut 2 12

3 Évaluation de comportements d'agents **15**
3.1 Problématiques et types de jeux 15
3.2 Conception de tests pour l'évaluation 17
3.3 Évaluer l'efficacité ou la crédibilité ? 17
3.3.1 Exemple d'évaluation de l'efficacité 18
3.3.2 Exemple d'évaluation de la crédibilité 18
3.3.3 Critiques 19

4 Proposition d'une nouvelle approche d'évaluation des comportements crédibles **20**
4.1 Présentation et objectifs 20
4.2 Protocole 21
4.3 Surveillance des personnages virtuels 22

5 Développement d'outils de surveillance des personnages virtuels d'Unreal Tournament 2004 **23**
5.1 Modifications apportées à Pogamut 2 23
5.1.1 Modification de Gamebots 24
5.1.2 Modification de Pogamut 2 25
5.2 La surveillance dans Pogamut 2 25

6 Application : exemple d'évaluation **27**
6.1 Protocole de l'expérience 27
6.1.1 Signatures 27
6.1.2 Distances entre signatures 28
6.1.3 Détermination des signatures des joueurs humains 31
6.1.4 Détermination des signatures des agents 32
6.2 Résultats pour les agents d'Unreal Tournament 2004 33

7 Conclusion **34**

A Résultats obtenus lors de l'expérience test **37**

1 Introduction

Dans le domaine de l'intelligence artificielle, de nombreuses recherches visent à concevoir et à développer des entités autonomes (agents) synthétiques dont le comportement serait comparable à celui d'un animal [Wilson 91] ou d'un humain. Le principe d'un agent est de percevoir des informations provenant de son environnement et, pour être qualifié d'intelligent, d'agir sur celui-ci de manière appropriée. Pour valider leurs recherches, les chercheurs ont donc besoin d'intégrer leurs développements dans un environnement pour les éprouver et les évaluer.

Cette intégration peut être très coûteuse en temps comme en argent lorsque qu'elle se fait dans le monde réel. Il faut en effet concevoir un robot et utiliser des techniques de traitement sur les données récupérées par les capteurs. Ces techniques sont souvent encore des sujets de recherche importants comme le traitement de données vidéo par exemple. Cela ajoute de nombreuses difficultés, principalement du fait que les informations tirées de ces capteurs ne sont pas fiables à 100%. L'intégration dans des environnements simulés est généralement bien plus aisée puisque l'espace, le temps et les interactions entre l'agent et l'environnement sont discrétisés.

Il existe de nombreux environnements simulés peu onéreux et de bonne qualité offerts par le monde de l'entreprise : les jeux vidéo. Les éditeurs de jeux cherchent en effet à immerger les joueurs dans des simulations qui se veulent au plus proche de la réalité, par le biais d'environnements simulés riches et complexes. Les chercheurs peuvent donc s'affranchir de certaines difficultés techniques (rendu, gestion de la physique, etc.) en utilisant des jeux pour mettre en situation les comportements d'agents qu'ils souhaitent étudier. Il est en effet très complexe et coûteux en temps de développer et de maintenir un environnement virtuel [Laird 02a; Robillard 03; Smith 00; Smith 01] car cela demande des compétences dans de nombreux domaines.

Entre autres avantages, les jeux vidéo sont prévus pour des humains, ils offrent donc un réel défi pour les agents et permettent aux chercheurs de déterminer si leurs développements adoptent des comportements comparables à ceux d'humains. Ils présentent en outre d'autres avantages [Laird 00] comme, par exemple, une communauté d'experts de ces environnements en la personne des joueurs, il est donc facile d'avoir des critiques pertinentes sur les agents créés. Enfin un point qui nous semble intéressant de souligner est que les chercheurs ne contrôlent pas totalement l'environnement dans lequel leurs agents sont évalués. Nous pensons que cela peut diminuer le biais introduit par l'habituel contrôle total et absolu de l'ensemble de l'évaluation, environnement virtuel inclus.

Ces nombreux avantages sont déjà reconnus et exploités dans la communauté des chercheurs. Les jeux vidéo sont en effet déjà utilisés dans de nombreux travaux de recherche, notamment pour les intelligences artificielles du niveau d'un humain [Laird 02a; Laird 02b], les histoires interactives [Cavazza 03; Mac Namee 04], la thérapie des phobies [Robillard 03], les musées virtuels [Lepouras 04] ou encore le test de modèles de comportements [Silverman 06]. On note par

ailleurs que cette tendance est assez nouvelle puisque la majorité des articles cités sont parus après 2000.

Cependant ce nombre important de travaux ne doit pas occulter le fait que l'utilisation des jeux vidéo n'est pas des plus simple. La très grande majorité des jeux disposent bien souvent d'un moteur d'intelligence artificielle gérant les comportements des agents. Même s'il est possible d'apporter des modifications mineures à ces comportements, il est cependant difficile de les modifier en profondeur, le code source n'étant pas disponible ou le langage dans lequel est codé le jeu étant trop limité pour les besoins. De plus, les jeux ne sont pas prévus pour évaluer les comportements d'agents, ceci reste donc à la charge du chercheur.

Nous nous sommes fixé comme objectif de passer en revue un certain nombre de solutions facilitant l'intégration et/ou l'évaluation de comportements d'agents dans les jeux. Selon les résultats de cette étude préliminaires, l'objectif était soit de créer de toute pièce des outils pour l'intégration et l'évaluation soit d'améliorer les points qui nous semblaient perfectibles dans la ou les solutions qui nous semblaient intéressantes.

L'objectif de ce rapport est donc d'exposer les méthodes possibles pour simplifier l'intégration (section 2) et l'évaluation (section 3) de nouveaux comportements d'agents dans les jeux vidéo. Nous mettrons en avant les avantages et inconvénients de ces méthodes pour ensuite proposer des solutions complémentaires notamment en ce qui concerne l'évaluation de comportement dit crédibles (section 4). Nous présenterons les développements effectué lors de nos travaux (section 5) pour ensuite présenter un exemple d'évaluation pour expliquer et valider nos travaux (section 6). Enfin nous ferons le bilan et proposerons des idées quant aux évolutions possibles (section 7).

2 Intégration de comportements d'agents aux jeux vidéo

Dans cette section, nous allons présenter les enjeux et les solutions pour l'intégration (sous-section 2.1) pour ensuite présenter nos travaux quant à l'amélioration de l'existant (sous-section 2.2).

2.1 Présentation des problématiques et état de l'art

2.1.1 Gestion des échanges entre le jeu et l'agent

Le principe des échanges entre l'agent et le jeu repose sur un échange perceptions/actions comme le montre la figure 1. Les perceptions peuvent être très variées comme l'état de fatigue du personnage virtuel que contrôle l'agent ou bien encore s'il pleut ou non dans le monde virtuel, etc. Il en va de même pour les actions qui peuvent être, si on suit l'exemple des perceptions, se reposer ou se déplacer en vue de se mettre à l'abri.

Le rôle du jeu vidéo est de recevoir les actions de tous les joueurs, qu'ils soient humains ou contrôlés par ordinateur, et de renvoyer des données symboliques et numériques représentant les perceptions à ces mêmes joueurs. Ces perceptions doivent bien évidemment rendre compte aux joueurs des actions qu'ils effectuent mais aussi des modifications de l'environnement virtuel, qu'elles soient causées par les actions des joueurs ou bien par des éléments dynamiques.

Le rôle de l'agent est quant à lui de récupérer les perceptions fournies par le jeu et de renvoyer des actions qu'il compte effectuer. Ces actions sont déterminées grâce aux perceptions mais aussi, bien souvent, grâce à des objectifs, à une expérience acquise, etc. L'objectif de l'intelligence artificielle est justement d'activer des actions qui sont appropriées en fonction de l'état dans lequel se trouve l'agent. On parle donc bien souvent du cycle perception-decision-action qui exprime l'enchaînement qui apparaît lorsqu'un agent évolue dans un environnement.

Le vocabulaire des perceptions et des actions en plus d'être très varié est bien souvent spécifique à chaque jeu. Par exemple dans un jeu où l'on contrôle un personnage qui se déplace en marchant et courant, on n'utilise que des déplacements en deux dimensions, alors que si l'on contrôle un vaisseau spatial, il est nécessaire de gérer des déplacements en trois dimensions. Il faut donc modifier le programme qui gère l'agent pour chaque jeu afin qu'il puisse comprendre les perceptions et envoyer des commandes d'actions compréhensibles par le jeu.

Pour que l'agent utilise au mieux ce vocabulaire, il doit aussi connaître les règles du jeu qui concernent directement celui-ci. Par exemple deux actions peuvent s'exclure mutuellement ou bien encore certaines actions durent plus ou moins longtemps et sont interruptibles ou non. Là aussi ces connaissances sont strictement spécifiques au jeu car elles dérivent des règles. Le code spécifique au jeu se retrouve donc dans la partie du programme qui gère les échanges perceptions/actions mais aussi dans les connaissances que l'agent a du jeu. Il en résulte qu'il est assez coûteux d'adapter des développements prévus pour un jeu pour un autre jeu.

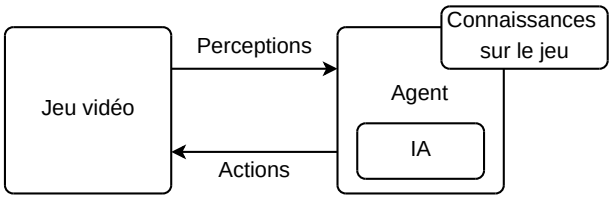


FIG. 1 – Principe de l'échange d'information entre le jeu et l'agent. Le sens des flèches indique le sens de transfert des données.

2.1.2 Approches client et serveur

Nous allons maintenant aborder le problème de l'intégration de manière plus technique en nous intéressant en particulier aux jeux en réseau. Dans ce type

de jeu, joueurs humains et des agents sont bien souvent mélangés dans la même partie, ce qui peut s'avérer très intéressant pour étudier le comportement des agents notamment en ce qui concerne la coopération. De plus, les agents jouent souvent le même rôle que les joueurs, ce qui est très avantageux lorsque l'on cherche à concevoir des agents dont le comportement serait similaire à celui d'un humain. Dans les jeux en réseau, un serveur crée une partie que des clients (les joueurs) peuvent rejoindre. Le serveur gère les échanges entre les clients mais il permet aussi de gérer les joueurs contrôlés par ordinateur qui sont fournis par défaut dans le jeu.

Un agent intégré à un serveur a plusieurs avantages que n'ont pas ceux intégrés à un client et inversement. Si un agent est intégré au serveur, il a le possibilité d'être omniscient car le serveur a accès à toutes les informations sur la partie qu'il gère. Cela peut être utile dans le cas où l'intelligence artificielle doit posséder des informations sur l'environnement en entier pour agir. Les agents qui n'ont besoin que d'une information locale peuvent aussi utiliser cette solution mais les informations disponibles ne reflétant pas celles dont dispose un joueur il n'est pas possible de garantir l'« honnêteté » de l'agent. En outre, comme le serveur est prévu pour utiliser des joueurs contrôlés par ordinateur, l'intégration est souvent plus facile à réaliser.

Une autre solution, qui ne nécessite aucune modification sur le serveur, consiste à créer un programme pour contrôler un agent capable de se faire passer pour un client classique (celui utilisé par les joueurs humains) auprès du serveur. Cela peut s'avérer très difficile car les éditeurs de jeu divulguent rarement le protocole qui permet les échanges de données sur le réseau pour limiter les cas de tricherie. Cette approche a l'avantage de permettre à l'agent de se connecter sur n'importe quel serveur, notamment un serveur créé et utilisé par des humains. Il est donc possible de confronter les agents aux humains sans que ces derniers ne sachent s'ils jouent contre des joueurs virtuels ou d'autres joueurs. Un autre point intéressant est que les agents n'ont qu'une vue locale ce qui n'est pas gênant pour la majorité des cas et est même un avantage puisque cela garantit qu'ils ne « trichent » pas. Cela exclu néanmoins l'intégration d'intelligences artificielles qui nécessitent d'être omniscientes.

Il existe une dernière possibilité, hybride de l'approche client et de l'approche serveur. Elle consiste à modifier le serveur pour que les agents puissent s'y connecter via le réseau. Ces agents ont bien souvent la possibilité d'être omniscients mais cette approche permet une plus grande flexibilité. Les programmes gérant le comportement des agents peuvent en effet s'exécuter sur des machines différentes ce qui peut permettre de répartir la charge de calcul. En outre, cela permet une grande flexibilité au niveau du langage de programmation utilisé puisqu'il suffit qu'il soit capable de gérer les échanges sur le réseau. La principale distinction avec l'approche client est que l'agent ne se connecte pas comme le ferait un joueur.

Pour de petits développements, il est souvent plus avantageux de modifier la gestion de l'intelligence artificielle sur l'application serveur car cela est moins coûteux en temps. Cependant, l'approche client est l'approche la plus séduisante

	Serveur	Client	Hybride
Omniscient	Oui	Non	Oui
Intégration	Simple	Complexe	Assez complexe
Répartition des calculs	Non	Oui	Oui
Modification du serveur	Oui	Non	Oui
Agents pouvant « tricher »	Oui	Non	Oui
Mélange différents agents	Non	Oui	Oui

FIG. 2 – Résumé des caractéristiques des approches serveur et client dans le cas général. « Répartition des calculs » indique s'il est possible de faire fonctionner les agents et le jeu sur différentes machines et « Mélange différents agents » si chaque agent peut être contrôlé par un programme différent.

car elle offre plus de possibilités (cf. figure 2) et les intelligences artificielles ont rarement besoin d'être omniscientes. Enfin, l'approche hybride est un bon compromis puisqu'elle est flexible même si elle ne permet pas d'intégrer des agents à n'importe quel serveur.

2.1.3 Solutions existantes pour l'intégration

Une fois l'approche choisie, il faut passer à l'intégration en elle-même. La solution la plus directe est d'utiliser les outils fournis par le jeu utilisé. Dans le cas d'Unreal Tournament[1] par exemple, le jeu définit un langage appelé UnrealScript qui permet de modifier le comportement des éléments dynamiques, et notamment celui des personnages contrôlés par ordinateur. Cependant ces outils ont leurs limites, pour suivre le même exemple, UnrealScript est un langage de description de machines d'état finis, il n'est donc pas adapté à d'autres approches dans la gestion des comportements d'un agent comme par exemple la programmation logique [Bratko 01] ou par contraintes.

Des outils commerciaux existent [XAI 08; AII 08] qui aident au développement d'intelligences artificielles dans les jeux. La définition des comportements est très simple, grâce à des environnements de développement conçu spécifiquement à cet effet. Cependant ces outils ne correspondent généralement pas aux besoins des chercheurs puisque, premièrement, le modèle de comportement est déjà établi, or le but des travaux de recherche est assez souvent de définir un nouveau modèle. Deuxièmement, ces outils ne permettent pas d'intégrer les agents dans des environnements virtuels mais uniquement de les développer.

Une autre solution offrant plus de flexibilité, est l'utilisation d'interfaces logicielles qui sont spécifiques à un jeu. Elles permettent de redéfinir le comportement des joueurs contrôlés par ordinateur dans le jeu sans utiliser le moteur de gestion de l'intelligence artificielle déjà présent dans celui-ci. Cependant ces interfaces sont peu nombreuses et sont quasi exclusivement développées pour les jeux de tir subjectif, limitant les problématiques couvertes. Nous allons donc étudier rapidement les solutions que nous avons jugées les plus intéressantes et qui sont bien souvent développées par des équipes de recherche.

[1] Jeu vidéo de tir subjectif développé par Epic Games et Digital Extremes, sorti en 1999.

FlexBot FlexBot [Fle 08] est une modification du jeu Half-Life[2] qui permet le développement aisé d'agents dans le jeu. Son objectif principal est de fournir une première expérience dans le monde de l'intelligence artificielle pour les étudiants.

QuakeBot QuakeBot [Qua 08] pour le jeu Quake[3] permet de connecter des agents via un réseau en utilisant le protocole défini par les développeurs du jeu. Cette interface n'est pas très utilisé en recherche mais elle offre une approche assez différente des autres. Quake étant un jeu assez vieux, cette interface à perdu de son intérêt.

F.E.A.R F.E.A.R [FEA 08] est une plate-forme de développement d'agents basée sur le jeu Quake 2[4]. Les agents F.E.A.R doivent être écrits en C++ et, pour quelques parties, en XML.

Gamebots Gamebots [Kaminka 02] permet de contrôler des agents dans le jeu Unreal Tournament en définissant un protocole réseau. Son objectif principal est de permettre aux agents de cohabiter avec des joueurs dans un environnement virtuel, permettant de développer des stratégies collaboratives par exemple. La conception de Gamebots est très flexible puisqu'il définit un protocole clair et documenté sur les échanges possibles entre l'agent et le jeu.

JavaBots Le projet JavaBots [Adobbati 01; Jav 08] met à disposition une bibliothèque Java pour communiquer avec Gamebots. Un programme très basique est fourni avec afin de connecter les agents au jeu et de les visualiser sur une carte de l'environnement.

Quagents Quagents [Brown 04] fournit à la fois un protocole réseau et une API[5] pour le jeu Quake 2. Cette interface a été conçue pour des étudiants, comme support à un cours d'intelligence artificielle.

TIELT TIELT [Molineaux 05] est une plate-forme pour l'apprentissage, prévu pour connecter un agent conçu pour apprendre à un moteur de jeu. L'objectif est de fournir un moyen pour les agents apprenant de travailler dans différents environnements.

Pogamut 2 Pogamut 2 [Burkert 07] est une plate-forme pour le développement rapide de comportements d'agents pour le jeu Unreal Tournament 2004[6]. Elle comprend un environnement de développement complet pour l'implémentation et l'évaluation d'agents. Cette plate-forme est le successeur de Pogamut 1 qui avait à peu près les mêmes objectifs mais pour le jeu Unreal Tournament.

[2] Jeu vidéo de tir subjectif développé par Valve Software, sorti en 1998.
[3] Jeu vidéo de tir subjectif développé par id Software, sorti en 1996.
[4] Jeu vidéo de tir subjectif développé par id Software, sorti en 1997.
[5] *Application Programming Interface*, interface de code source fournie par un système informatique ou une bibliothèque logicielle, en vue de répondre à des requêtes pour des services qu'un programme informatique pourrait lui faire.
[6] Jeu vidéo de tir subjectif développé par Epic Games et Digital Extremes, sorti en 2004.

Pogamut 2 nous a semblé la solution la plus intéressante de celles que nous venons de présenter. L'intérêt principal de cette plate-forme est qu'elle est maintenue par une équipe très dynamique (le projet est mis à jour plusieurs fois par semaine) et compétente puisqu'ayant déjà développé Pogamut 1. De plus plusieurs documentations sont disponibles et mises à jour et l'équipe est très réactive aux questions qui lui sont posées. Enfin, Pogamut 2 est la seule solution qui offre à la fois :

- Une extensibilité et modularité suffisante pour se connecter à un grand nombre de programmes modélisant des comportements d'agents ;
- Un environnement de développement facile à utiliser permettant de réaliser l'implémentation, le déboguage et des expériences ;
- Une parallélisation du programme gérant l'environnement virtuel (Unreal Tournament 2004) et du programme modélisant les comportements d'agents ;
- Une approche qui simplifie grandement le développement d'agents sans pour autant limiter la complexité des comportements qui peuvent être codés ;
- Des outils pour l'évaluation.

Pogamut 2 répond donc en grande partie aux problématiques d'intégration et d'évaluation. Nous allons maintenant détailler le fonctionnement de Pogamut 2 puisque c'est cette plate-forme que nous avons utilisé tout au long du stage.

2.1.4 Pogamut 2

Pour comprendre l'intérêt des différents composants de Pogamut 2 nous allons expliquer son architecture en partant de la figure 1.

Unreal Tournament 2004 Tout d'abord, Pogamut 2 est basé sur le jeu Unreal Tournament 2004 (UT2004). Il a été choisi pour développer la plate-forme Pogamut 2 car c'est un jeu très populaire parmi les joueurs. Ce jeu est codé en partie en C++ et en partie en UnrealScript, langage spécifiquement développé pour le jeu. La majorité du code C++ ne peut pas être obtenue gratuitement, mais elle gère uniquement le noyau du jeu (rendu, réseau, etc.), parties qui ne nous intéressent pas pour modifier les comportements d'agents. UnrealScript quant à lui qui permet de définir simplement le comportement du jeu : règles du jeu, comportement des éléments dynamiques (incluant les agents), etc. toute cette partie étant *open-source*. C'est pour cette raison que beaucoup de modifications du code UnrealScript existent, crées par des joueurs. Ces mêmes joueurs ont par ailleurs mis sur internet un grand nombre d'aides et documentations pour aider à la modification du jeu. Un autre avantage est qu'il est possible de rendre compatible du code UnrealScript d'une version d'Unreal Tournament à une autre, moyennant quelques modifications. Enfin, un atout majeur est qu'Unreal Tournament 2004 fonctionne sous Windows, Linux et Mac ce qui en fait un outil très flexible.

Pour ceux qui ne sont pas familiers avec les jeux vidéo et notamment Unreal Tournament, une petite parenthèse s'impose pour en expliquer les mécanismes de jeu. Unreal Tournament 2004 est un jeu de tir subjectif, plus communément appelé *First Person Shooter* (FPS), en d'autres mots, chaque joueur ou agent contrôle un unique personnage virtuel et voit « par ses yeux ». Le personnage

peut, non exhaustivement, récupérer des objets (armes, bonus, ...), se déplacer (marcher, sauter, ...) et tirer avec une arme. Chaque personnage possède un nombre de points de vie caractérisant sa santé : chaque fois qu'un personnage est touché par un tir ennemi, une certaine quantité de points de vie est retirée à la valeur actuelle. Quand les points de vie atteignent zéro, le personnage « meurt » mais réapparaît habituellement à un autre endroit dans l'environnement. Même si le concept peut paraître simpliste, il peut s'avérer être un défi pour les intelligences artificielles quand d'autres règles sont ajoutées en plus des règles basiques. Les agents doivent alors mettre en place des stratégies en plus d'avoir des aptitudes au combat pour réussir à battre les humains.

Unreal Tournament permet donc de gérer l'environnement virtuel, les évènements qui s'y déroulent et les personnages virtuels (potentiellement des agents) qui y évoluent comme le montre le figure 3.

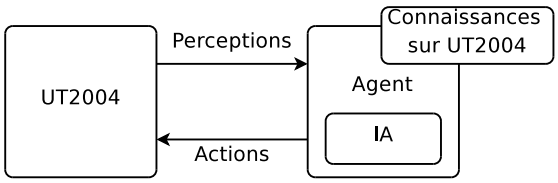


FIG. 3 – Principe de l'intégration d'un agent à Unreal Tournament 2004.

Gamebots 2004 Comme on l'a évoqué plus tôt, UnrealScript qui permet entre autre de coder les comportements d'agents dans Unreal Tournament 2004, a quelques limitations. De plus, les chercheurs ne devraient pas a avoir à apprendre un nouveau langage de script pour intégrer des comportements à un jeu. C'est pour cela que la plate-forme Pogamut 2 intègre une version modifiée de Gamebots. Celle-ci à été rendue compatible avec la version 2004 du jeu grâce à quelques modifications. Cela permet d'exporter les perceptions et de récupérer les actions d'un agent via le réseau (cf. figure 4). Le principe est très simple : lorsque certaines méthodes concernant le personnage virtuel sont appelées dans le jeu, Gamebots envoie les messages correspondants sur le réseau. Il est aussi à l'écoute de commandes envoyées par l'intelligence artificielle. D'autres fonctionnalités très utiles ont été intégrées à cette nouvelle version, baptisée Gamebots 2004, telles que la possibilité de faire un lancer de rayon et de récupérer les caractéristiques de l'objet percuté par ce rayon. Les messages texte correspondant aux actions et perceptions définissent un protocole.

Parser Le problème de Gamebots est que la gestion des échanges de messages texte est extrêmement fastidieuse. Pogamut 2 intègre donc un module appelé parser qui récupère les messages envoyés par Gamebots et crée des objets Java contenant les informations reçues (cf. figure 5). Ces objets Java sont bien plus aisés à manipuler que des chaînes de caractères. Le parser est aussi capable d'envoyer vers Gamebots les commandes qui lui sont demandées par l'agent. Notons que même si la traduction de *parser* est analyseur syntaxique, ce module

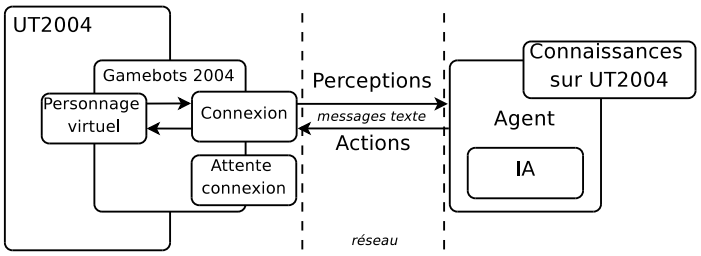


FIG. 4 – Principe de l'intégration d'un agent à Unreal Tournament 2004 grâce à l'utilisation de Gamebots. L'intérêt de Gamebots est de pouvoir contrôler un personnage virtuel dans Unreal Tournament 2004 via le réseau.

ne se limite pas seulement à cette tache puisque sa fonction est plus proche d'un compilateur.

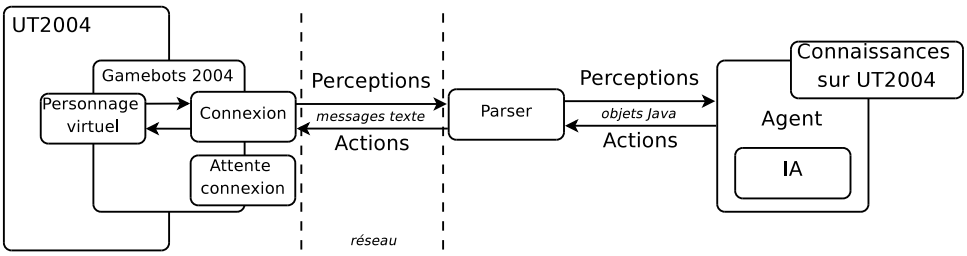


FIG. 5 – Principe de l'intégration d'un agent à Unreal Tournament 2004 grâce à l'utilisation de Gamebots et d'un module parser. L'intérêt de ce dernier est de centraliser la gestion des messages échangés avec Gamebots. Ce parser est et doit être compatible avec le protocole défini par Gamebots.

Client Le client est un ensemble de structures de données correspondant à la mémoire de l'agent et permet d'exécuter simplement des actions grâce à des méthodes Java appropriées. Il intègre déjà toutes les connaissances des base sur le jeu (cf. figure 6), nous verrons plus en détail son architecture par la suite.

IA Les comportements sont codé par les chercheurs, il permettent, en fonction des information contenues dans le client de prendre des décisions et d'appeler des méthodes correspondant aux actions que l'agent désire effectuer. C'est uniquement cette partie que doit coder les chercheurs qui veulent utiliser Pogamut 2.

Environnement de développement intégré (IDE) L'IDE fournit un ensemble de fonctionnalités permettant de faciliter le développement de comportements : gestion du serveur, déboguage des agents, création d'expériences, etc.

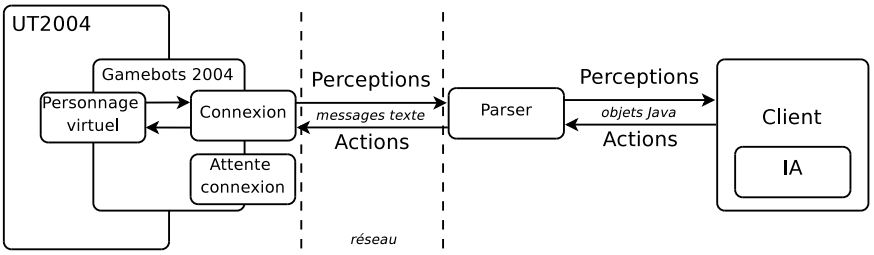


FIG. 6 – Principe de l'intégration d'un agent à Unreal Tournament 2004 grâce à l'utilisation de Gamebots, d'un module parser et du module client de Pogamut 2. L'intérêt de ce dernier est de mémoriser les informations qui lui parviennent du parser et de mettre à disposition des méthodes Java permettant de faire effectuer des actions au personnage virtuel.

Il est développé sur la forme d'un *plugin* NetBeans[7] afin de limiter le temps et la complexité des développements. L'ensemble de l'architecture de Pogamut 2 est résumé par la figure 7.

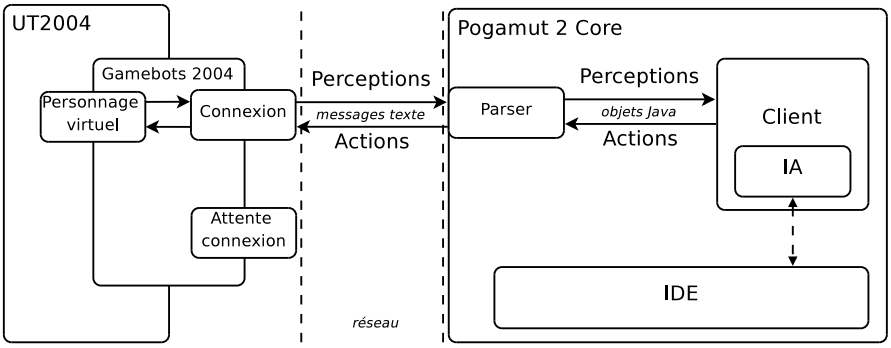


FIG. 7 – Architecture générale de Pogamut 2. L'intérêt de l'IDE est de faciliter le développement de comportements grâce à de nombreux outils.

Nous allons maintenant détailler l'architecture du client. Cette partie est très importante car c'est uniquement avec cette partie que dialogue l'intelligence artificielle développée par les chercheurs.

Corps Le corps est la partie la plus importante du client puisque c'est le seul composant qui dialogue avec le parser. Toutes les perceptions et actions passent par lui. Les perceptions sont redistribuées entre les structures de données appropriées. Les actions quant à elles sont directement reçues du programme gérant la prise de décision de l'agent.

[7] Plate-forme de développement d'applications graphiques Java développé par Sun Microsystems.

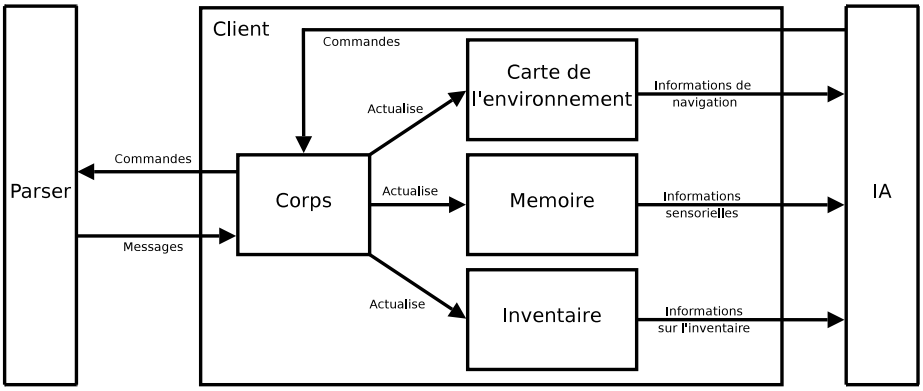


FIG. 8 – Architecture du client dans Pogamut 2. Le parser et l'intelligence artificielle on été ajoutés pour plus de clarté.

Connaissances La carte de l'environnement, la mémoire et l'inventaire constituent les connaissances auxquelles a accès l'intelligence artificielle. La carte regroupe des informations sur la position de tous les objets statiques, comme les bonus à ramasser au sol par exemple, ainsi que le graphe permettant de naviguer dans l'environnement. La mémoire contient les informations de tous les senseurs, à savoir les objets et personnages visibles, les bruits entendus, etc. Enfin l'inventaire regroupe toutes les données sur ce que possède l'agent, principalement composé d'armes (cf. règles du jeu expliquées précédemment).

2.1.5 Un agent dans Pogamut 2

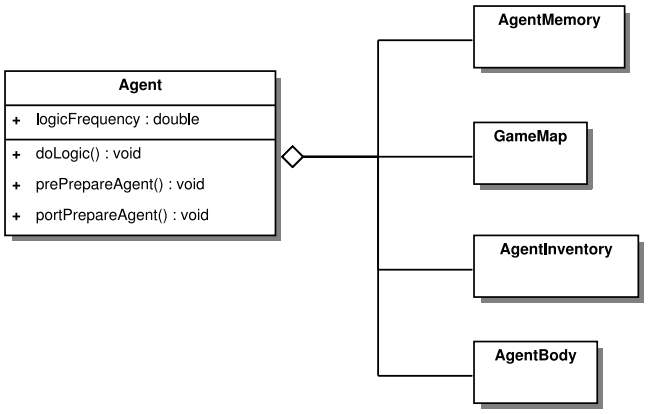


FIG. 9 – Diagramme de classes UML simplifié d'un agent dans Pogamut 2.

La création d'un agent est très simple. Comme le montre la figure 9, une classe `Agent` regroupe le corps est les connaissances tels qu'ils ont été défini précédemment. Il suffit alors d'hériter de cette classe pour créer un agent prêt à l'emploi,

ayant accès à toutes les méthodes dont il pourrait avoir besoin. La méthode `doLogic` est appelée à une fréquence de `logicFrequency`. Pour coder les prises de décision de l'agent, il convient donc de redéfinir cette méthode avec le code correspondant aux comportements de l'agent. L'attribut `logicFrequency` peut être redéfini dans les limites du raisonnable (notion qui dépend principalement de la puissance de la machine) pour augmenter ou diminuer le nombre d'appels à `doLogic` par seconde.

Les méthodes `prePrepareAgent` et `postPrepareAgent` peuvent elles aussi être redéfinies pour initialiser des variables avant le début des appels à `doLogic`. Cela peut-être très utile pour initialiser un réseaux de neurones par exemple. La différence entre les deux méthodes est le moment où elle sont appelées, `postPrepareAgent` ayant plus d'informations sur l'agent que `prePrepareAgent`. Si l'agent a besoin de relâcher des ressources à la fin de son exécution, il existe bien entendu la méthode `finalize` de Java prévue à cet effet.

Bien sûr, il est aussi possible de coder dans un autre langage de programmation que Java, Pogamut 2 étant compatible avec le langage POSH [Bryson 01]. POSH étant conçu pour coder la prise de décision d'agents, il est un outil particulièrement intéressant dans notre cadre de recherche. Il est aussi théoriquement possible de coder dans de nombreux autre langages de programmation, majoritairement des langages de script (python, etc.), mais POSH et Java offrent déjà des solutions simples et complètes.

2.2 Étude et travaux sur l'interopérabilité de Pogamut 2

Comme on l'a évoqué précédemment, la principale limitation des interfaces d'intégration actuelles est qu'elles sont développées uniquement pour un seul jeu. La compatibilité avec de nombreux jeux est un avantage certain, car, comme nous le verrons en détail en sous-section 3.1, un grand nombre de jeux différents permet de couvrir un grand nombre de problématiques. Les chercheurs peuvent donc trouver le jeu approprié pour développer le type de comportement désiré dans la liste des jeux compatibles. Il serait donc très intéressant de fournir un développement unique qui pourrait intégrer des agents dans différents types de jeux.

Nous allons maintenant étudier la composition d'un jeu pour mieux comprendre comment il serait possible de le modifier en vu d'intégrer les développements de chercheurs. Le composant permettant la définition d'un environnement virtuel dans un jeu est le moteur de jeu [Trenholme 08]. C'est le composant central des jeux vidéo, il fournit les technologies nécessaires à la création d'un jeu, ce qui simplifie grandement le développement de nouveaux jeux, c'est un *middleware*[8]. Beaucoup de moteurs existent sur le marché et plusieurs jeux sont développés sur chacun de ces moteurs.

Si une interface dialogue directement avec le moteur il peut être possible d'avoir des développements qui peuvent servir à tous les jeux basés sur ce moteur

[8] Logiciel qui fournit des fonctions qui restent personnalisables et qui évitent donc de « réinventer la roue ».

(cf. figure 10 à gauche). Un tel type d'interface est très polyvalent puisqu'elle est compatible avec un nombre important de jeux, par exemple l'Unreal Engine 2 est utilisé par plus de 50 produits commerciaux. Il faut cependant noter que les jeux basés sur un moteur sont souvent du même type puisque les moteurs offrent des outils qui sont orientés pour le développement d'un type de jeu. Bien sûr il est quand même plus avantageux de couvrir plusieurs types de jeu proches, une interface basée sur un moteur a donc un avantage certain en terme d'interopérabilité.

Une autre approche (cf. figure 10 à droite) est de créer un noyau qui regroupe des fonctions d'évaluation et de gestion des échanges perceptions/actions auquel il est possible d'y ajouter des modules spécifiques à chaque jeu. Cela demande plus de travail pour intégrer un agent dans un nouveau jeu que la solution proposée précédemment, mais elle offre une très grande flexibilité tout en garantissant une interface et des outils d'évaluation communs pour tous les jeux.

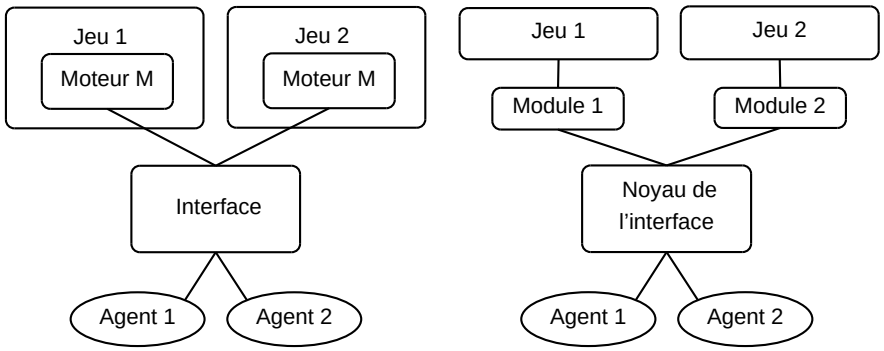


FIG. 10 – Conceptions envisageables pour une interface d'intégration et d'évaluation polyvalente. À gauche, l'interface interagit directement avec le moteur de jeu, à droite des modules permettre de gérer les spécificités de chaque jeu.

Lors de notre stage, nous avons essayé d'appliquer un des deux principes que nous avons énoncé ou une combinaison de ceux-ci afin de rendre Pogamut 2 plus complet au niveau des environnements qu'il propose. Il existe différents niveaux où il est possible de modifier le comportement de l'environnement. Tout d'abord en changeant de jeu comme il a été expliqué précédemment ou bien en modifiant les règles du jeu plus ou moins profondément.

Notre première tâche à été de déterminer si Pogamut 2 pouvait être rendu compatible avec d'autres jeux basés sur le même moteur de jeu qu'Unreal Tournament 2004. Nous avons utilisé pour nos tests le jeu America's Armies[9] qui a pour principal avantage d'être gratuit et en libre téléchargement. La phase critique de la modification de Pogamut 2 est l'intégration de Gamebots 2004 au nouveau jeu. Cela a été un échec dans le cas d'America's Army pour la

[9] Jeu vidéo de tir tactique multijoueur en vue subjective. Développé par les forces armées des États-Unis il a pour but d'améliorer l'image de l'US Army et d'inciter les gens à s'enrôler.

simple raison que l'armée américaine n'a pas rendu la partie codée en UnrealScript *open-source*. Cependant il est tout à fait possible théoriquement d'adapter Gamebots à d'autres jeux qui fournissent leur code UnrealScript comme par exemple le jeu Deus Ex : Invisible War. Malheureusement, par manque de temps nous n'avons pas pu vérifier cette théorie. Nous pouvons donc seulement affirmer que Pogamut 2 n'est pas compatible avec tous les jeux basés sur le moteur d'Unreal Tournament 2004 mais cela n'exclut pas qu'il soit compatible avec certains.

Une solution moins radicale que le changement de jeu est d'utiliser ce que les joueurs appellent des *mod*, version contractée du mot « modification ». Cela consiste à modifier les règles d'un jeu de façon plus ou moins importante. Parmi les modifications mineures on trouve l'ajout de nouvelles armes ou bien encore le changement de la gravité. Pour les modifications les plus importantes, certains joueurs ont transformé le jeu Unreal Tournament 2004 en un jeu de course de voiture ou bien en un jeu de balle (type football). Nous avons réussi à rendre Gamebots 2004 compatible avec ce genre de modification de l'environnement, cependant nous nous sommes rendu compte que les développements étaient assez complexes (mais pas impossible). En effet, de nouvelles règles du jeu impliquent de nouvelles perceptions et de nouvelles actions. Si l'on reprend les schémas de l'architecture (figures 7 et 8) on s'aperçoit qu'il est nécessaire de modifier toutes les parties entre Unreal Tournament 2004 et les comportement codés par le chercheur : Gamebots, le parser, le corps et les connaissances. Même si cela ne demande pas beaucoup de modifications, cela reste assez compliqué puisqu'il faut connaître à la fois UnrealScript pour Gamebots ainsi que le fonctionnement du parser et du client.

Enfin, le jeu original comporte différentes règles du jeu souvent appelé *gametypes*. Actuellement Pogamut 2 est compatible avec les règles *deathmatch* qui correspond aux règles basiques expliquées plus tôt, *team deathmatch* où les joueurs sont en équipes, *double domination* où les équipes doivent contrôler des points stratégiques et *capture the flag* où les équipes doivent ramener à leur base un drapeau se trouvant dans une base adverse. Cependant il existe encore d'autres règles du jeu dont certaines sont très intéressante de par les stratégies qu'elles demandent à mettre en place pour gagner. On retombe sur les mêmes problèmes que pour les modifications puisque les règles du jeu définissent des nouvelles actions et perceptions.

En conclusion, Pogamut 2 est polyvalent mais l'adaptation à de nouvelles règles est assez complexe. Cela n'est pas spécifique à Pogamut 2 puisque c'est le principe même des nouvelles règles qui oblige à une modification du code de tous les composants permettant le transit des actions et perceptions entre les comportements et le jeu. On pourrait reprocher à Pogamut de ne pas simplifier ces ajouts, mais à l'heure actuelle, des efforts sont fait dans ce sens. En particulier, les messages transitant de Gamebots à Pogamut 2 sont définis clairement grâce à un fichier XML qui génère automatique les objets Java correspondants.

3 Évaluation de comportements d'agents

Une fois l'intelligence artificielle intégrée au jeu, l'objectif est maintenant d'évaluer son comportement face aux différentes situations qui se présentent à elle et, plus généralement, sa capacité à résoudre des problèmes. Le but de cette mise en situation est de valider des recherches menées dans le domaine de l'intelligence artificielle en prouvant que les développements répondent aux problématiques. Cependant ces problématiques sont très nombreuses et il n'est pas évident d'évaluer si la réponse proposée est satisfaisante ou non.

3.1 Problématiques et types de jeux

Les jeux vidéo ont la caractéristique intéressante de définir des rôles qui sont assez variés et qui touchent un grand panel de problématiques du monde de l'intelligence artificielle. Ces rôles peuvent être des sujets assez complexes pour permettre le développement d'agents du niveau d'humains. Dans les jeux vidéo actuels, il est possible de définir, non exhaustivement, les rôles d'ennemis tactiques, de partenaires, de personnages secondaires, d'adversaires stratégiques, d'unités, de commentateurs ou de scénaristes [Laird 00]. La figure 11 résume ces rôles et les problématiques qu'ils posent.

Le rôle d'ennemi tactique est le plus utilisé dans le monde de la recherche, on les trouve dans les jeux où les joueurs et agents incarnent des personnages qui doivent s'affronter. C'est un combattant qui, en plus de savoir se déplacer [Schneider 03], se cacher ou attaquer doit pouvoir aussi mettre en place et suivre des tactiques. Il doit être totalement autonome et peut essayer de prévoir les actions de ses adversaires [Laird 01]. Si on cherche à faire coopérer plusieurs ennemis tactiques dans des équipes, on parle de partenaires. Ils doivent pouvoir mettre en place des tactiques de groupe ce qui demande de communiquer avec les autres agents et aussi avec les joueurs humains.

Dans les jeux d'aventure et les jeux de rôle, les joueurs parcourent un monde virtuel qui est peuplé de personnages secondaires eux aussi virtuels. Ces agents doivent être capables de discuter avec les joueurs mais aussi d'avoir des objectifs, des émotions. Il cherche donc à simuler les interactions avec un humain. Ils peuvent aussi avoir les mêmes besoins que les partenaires s'ils sont amenés à combattre. Ces agents sont donc très complets et essayent d'imiter au mieux le comportement d'un humain.

Dans les jeux de stratégie où il faut gérer des bâtiments et des unités de guerre, ou bien dans les jeux de sport où des entraîneurs déterminent la composition des équipes et le style de jeu, les agents peuvent jouer le rôle d'adversaires stratégiques. Ils doivent être capables d'établir des plans et contrer ceux de leurs adversaires en gérant au mieux leurs ressources et leurs unités. Le défi est de créer un agent qui est capable de planifier à court et à long terme ainsi que d'estimer l'évolution et les choix de ses adversaires. Les unités constituent aussi un rôle que peuvent jouer les agents, ils doivent avoir une réaction réaliste aussi bien individuellement que collectivement.

Un autre rôle assez différent des précédents est le commentateur. Comme dans la réalité, il commente les parties dans les jeux de sports mais toutes ses phrases sont générées automatiquement en fonction des actions sur le terrain. Cela permet de rendre plus vivant les jeux de sports et donne une impression de réalisme. Pour accomplir cette tâche l'agent doit à la fois avoir des connaissances sur le sport et surtout être capable de formuler des phrases correctes, sensées et adaptée à la situation.

Enfin, le dernier rôle, plus rare dans les jeux, est le rôle de scénariste. Il s'assure que l'histoire reste cohérente et intéressante en tenant compte des actions du joueur et en agissant sur l'environnement et les personnages qui le peuplent. Une autre application qui n'est pas réellement du domaine des jeux mais qui colle parfaitement au rôle de scénariste est de tourner une série virtuelle avec des acteurs virtuels dirigés par un scénariste et metteur en scène virtuels dans un jeu vidéo [Cavazza 03]. Dans les deux cas, l'agent doit avoir une connaissance de scénarios intéressants et de comment les amener subtilement grâce au jeu des acteurs et des événements dans l'environnement.

Type de jeu		**Problématiques**
Action		Interagir avec l'environnement
Jeux de rôles		Réponse rapide
Jeux d'aventure		Sensations réalistes
Jeux de stratégie		Adaptation à l'environnement
God games[10]		Interagir avec des humains
Sports d'équipe		Adaptation aux joueurs
Sports individuels		Ajustement de la difficulté
		Adaptation aux stratégies
		Interagir avec d'autres agents
⇓		Coordination de comportements
		Navigation
		Utilisation de stratégies et tactiques
		Gestion des ressources
Rôles des agents	⇒	Comprendre le cours du jeu
Ennemis tactiques		Réponses semblables aux humains
Partenaires		Temps de réaction
Personnages secondaires		Mouvements réalistes
Adversaires stratégiques		Émotions
Unités		Personnalités
Commentateurs		Faible coût en calculs
Scénaristes		Faible coût de développement

FIG. 11 – Résumé des principaux genres de jeux vidéo ainsi que des rôles et des problématiques couverts par ceux-ci. Traduit de [Laird 00].

Bien sur, les jeux n'ont pas tous un type clairement défini et il en va souvent de même pour les rôles possibles. Il est donc assez facile de trouver des jeux qui posent les problématiques qui nous intéressent et qui offrent un sujet d'étude

intéressant pour faire évoluer les techniques d'intelligence d'artificielle vers un niveau plus proche de celui d'un humain.

3.2 Conception de tests pour l'évaluation

Les problématiques couvertes par les différents rôles sont nombreuses, ce qui rend parfois difficile l'évaluation des intelligences artificielles. Il faut en effet établir un protocole de test pour chaque problème posé et évaluer si la réponse est satisfaisante. Pour un ennemi tactique par exemple, il faut déterminer s'il se déplace correctement dans l'environnement, s'il met en place des tactiques appropriées selon sa situation, etc. De plus, il n'est de plus pas toujours possible de mesurer de façon objective la qualité de la réponse puisque certains critères sont totalement subjectifs, comme les émotions par exemple.

Il est aussi intéressant d'évaluer l'influence des paramètres internes de l'agent sur ses résultats. Cela permet de voir quels sont les réglages prometteurs et ceux qui ne le sont pas ou bien encore de démontrer les avantages d'un nouveau développement. On s'aperçoit rapidement que le temps d'évaluation est très important, sachant qu'en plus ces tests se font très souvent en temps réel car le fait d'accélérer la vitesse de simulation modifie les résultats. Il est donc intéressant de combiner de nombreux tests en une seule séance pour réduire la durée totale de l'évaluation.

Outre le temps important d'évaluation, il faut apporter un soin tout particulier à la conception des tests pour ne pas fausser les résultats obtenus. Cette phase est particulièrement importante pour les jeux vidéo où les environnements sont très complexes et où beaucoup de paramètres peuvent influencer les résultats. Par exemple, dans le cas où deux types d'agents s'affrontent, il faut veiller, en règle générale, à ce que les deux types soient sur un pied d'égalité : le terrain ne doit pas avantager un agent au détriment de l'autre.

Enfin, une fois les résultats des tests obtenus, il faut les interpréter. Cette étape est spécifique à chaque test et chaque problématique. Néanmoins il est intéressant dans cette phase de se faire épauler par un expert du jeu, un joueur assidu. Sa bonne connaissance de l'environnement que représente le jeu permet en effet d'éclaircir certains résultats qui peuvent apparaître parfois étranges et apporter un point de vue totalement différent de celui des chercheurs.

3.3 Évaluer l'efficacité ou la crédibilité ?

Nous allons maintenant nous intéresser plus en détail aux agents dont le comportement est dit crédible. Une définition communément acceptée de la crédibilité est que « les utilisateurs qui interagissent avec l'agent pensent qu'ils observent un être pensant qui a ses propres croyances, des désirs et une véritable personnalité » [Lester 97]. Les recherches dans ce domaine sont nombreuses car c'est un facteur très important pour immerger un utilisateur dans un environnement virtuel (notion dite de *presence* [Heeter 92] en anglais) grâce aux

[10] Jeux vidéo de simulation où le joueur prend la position d'un dieu ou d'un meneur qui peut contrôler des unités autonomes.

intelligences artificielles. Il est donc intéressant de pouvoir évaluer la crédibilité, cependant étant une notion subjective, cette évaluation est très complexe.

Il est intéressant de noter que les agents crédibles ne poursuivent pas les mêmes buts que les agents réalistes et les agents intelligents, même s'ils partagent des points communs [Loyall 97]. Un agent crédible n'est pas forcement réaliste, preuve en est faite par les personnages de dessins animés par exemple qui sont tout à fait crédible mais pas réalistes. Un agent réaliste devrait être considéré comme crédible bien que la crédibilité étant totalement subjective, cette affirmation ne peut pas être vérifiée. Enfin, un agent intelligent n'est clairement pas crédible quand ses capacités dépassent de loin celle d'un humain, mais un agent crédible se doit d'être raisonnablement intelligent.

Puisque comportements crédibles et intelligents ont des points communs, une autre possibilité est de mesurer l'efficacité pure de l'agent grâce à des scores. Cette mesure est habituellement totalement objective car elle se base sur des fait mesurables sans l'intervention d'humains. Nous allons étudier plus précisément comment efficacité et crédibilité sont mesurées dans quelques travaux de recherche pour mieux comprendre les enjeux et les difficultés rencontrées.

3.3.1 Exemple d'évaluation de l'efficacité

Objectif L'objectif de cette évaluation est de mesurer l'efficacité des agents dans le jeu conformément aux scores qui servent habituellement à évaluer le gagnant des parties. Le système de notation est donc déjà intégré au jeu, il n'est donc besoin d'aucun développement supplémentaire.

Protocole Dans [Robert 05], l'évaluation se base principalement sur un affrontement entre une équipe d'agents développés dans les travaux de recherches contre des agents développés par des joueurs. L'intérêt principal de cette méthode est d'avoir un grand nombre de résultats. En effet, dans les tests menés, les parties durent 24 heures et celles-ci sont répétées cinq fois, les résultats obtenus sont donc assez représentatifs. Il est alors possible d'évaluer la qualité de l'agent en fonction des différentes modifications apportées à celui-ci comparé à des intelligences artificielles de niveau débutant ou confirmé — qualification établie par les joueurs eux-mêmes.

Résultats Un exemple de résultats est donné en figure 12 où les agents développés, les MHiCSbots sont confrontés aux Foxbots, déjà développés par des joueurs et ayant le niveau de joueurs confirmés. Le score est calculé très simplement : chaque équipe marque 10 points lorsqu'un membre ce celle-ci rapporte le drapeau ennemi dans sa propre base (règles dite du *capture the flag*).

3.3.2 Exemple d'évaluation de la crédibilité

Objectif L'objectif de l'évaluation est de déterminer si des agents ont un comportement crédible dans un environnement virtuel en d'autres termes s'ils donnent l'illusion que les personnages virtuels sont vivants de par leur comportement.

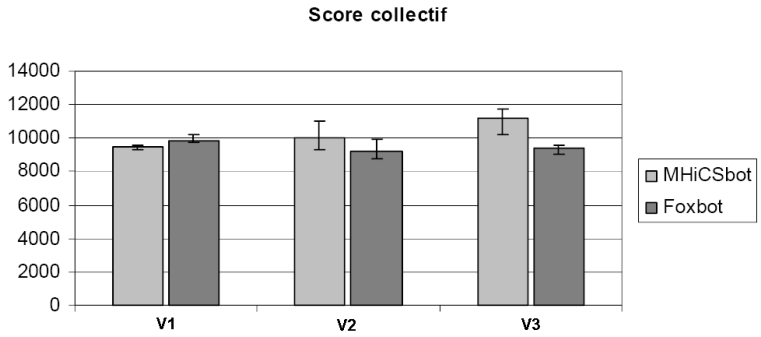


FIG. 12 – Exemple de résultats de tests obtenus dans [Robert 05] contre des agents déjà développés. Ils correspondent aux scores collectifs obtenus dans les parties opposant les MHiCSbots, résultats des recherches, aux Foxbots, programme développé par des joueurs. Les différents cas montrent les résultats obtenus avec trois versions des MHiCSbots (V1, V2 et V3).

Protocole Dans l'expérience menée dans [Mac Namee 04], deux simulations représentant un bar virtuel ainsi que des clients eux aussi virtuels ont été montrés à un groupe de 13 personnes. Dans l'une, les comportements des agents étaient déterminés de façon aléatoire, dans l'autre ils utilisaient l'architecture développée dans les travaux de recherche. Il est demandé aux 13 volontaires de remplir un questionnaire qui comprend une question à 2 choix, une question ouverte et 6 questions de notation.

Résultats Pour chaque question à choix multiple ou de notation, il est possible, en plus de graphiques montrant la répartition des réponses (cf. figure 13), de faire des tests d'hypothèses. Pour les résultats présentés ici, l'hypothèse selon laquelle la majorité des personnes trouvent l'architecture proposée la plus crédible à été validée avec une confiance de plus de 95%. Les résultats peuvent donc être considérés comme valables. Les questions ouvertes permettent seulement aux chercheurs d'avoir l'avis général des participants au test mais ne rentrent pas réellement en compte dans l'évaluation.

3.3.3 Critiques

L'évaluation de l'efficacité a des avantages très intéressants. En plus d'être assez simple à mettre en place, l'évaluation est à la fois objective et, dans la plupart des cas, ne demande pas l'intervention d'un humain. Cela rend possible le déroulement d'expériences sur de très longues durées pour diminuer l'écart-type des résultats. Il est aussi possible d'utiliser ces mesures pour les algorithmes d'optimisation, comme la fonction de *fitness* d'un algorithme génétique par exemple. Par contre ces mesures ne sont pas adaptées pour évaluer des comportements crédibles, elles sont plutôt utiles pour évaluer des comportements intelligents ou des agents ayant de bonnes compétences.

Pour évaluer la crédibilité, notion subjective, des humains sont nécessaires. Cela implique que les mesures sont beaucoup plus longues à obtenir, compara-

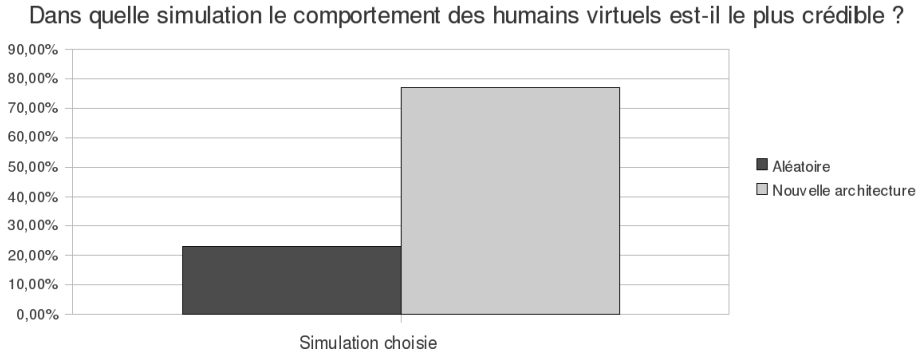


FIG. 13 – Exemple de résultat obtenu pour une expérience sur la crédibilité des agents dans les travaux de [Mac Namee 04]. D'autres résultats ont été tirés du questionnaire utilisé, étant du même type que celui présenté, nous ne les donnons pas.

tivement aux mesures d'efficacité. De plus selon certaines recherches, les questionnaires, qui sont habituellement utilisés dans ce genre d'évaluation, ne devraient pas être utilisés comme instruments de mesure mais plutôt pour émettre des hypothèses [Slater 04]. Malgré tous ces désavantages, c'est actuellement la meilleure méthode pour évaluer si des agents se comportent comme des humains le feraient.

Dans un certain nombre de recherches où la crédibilité n'est pas l'objectif principal [Le Hy 07] mais un atout certain, les deux types d'évaluations sont utilisés. L'efficacité permet, dans un premier temps, d'optimiser les comportements des agents, soit par un algorithme, soit manuellement. En suite, une fois les comportements correctement paramétrés, quelques évaluations de la crédibilité sont faites pour avoir une idée des possibilités du nouveau modèle proposé en ce qui concerne la crédibilité.

4 Proposition d'une nouvelle approche d'évaluation des comportements crédibles

4.1 Présentation et objectifs

Au vu des problèmes rencontrés pour l'évaluation de la crédibilité, nous nous sommes fixé pour objectif de définir une nouvelle méthode d'évaluation ayant les avantages des mesures d'efficacité et permettant d'évaluer des comportements crédibles. Le point le plus important de cette nouvelle approche est que nous n'allons pas évaluer la crédibilité puisque c'est la subjectivité de cette notion qui pose problème lors de l'évaluation. Nous allons donc chercher à mesurer objectivement la « proximité » entre un comportement artificiel et celui d'un humain. La principale limitation de cette nouvelle mesure est que agents et humains doivent pouvoir jouer le même rôle dans l'environnement virtuel pour pouvoir les comparer.

Comme nous en avons discuté précédemment, la notion de crédibilité, de réalisme et d'intelligence sont proches mais ne sont pas synonymes. La nouvelle mesure que nous allons présenter se situe entre réalisme et crédibilité. En effet, d'un coté nous cherchons à avoir des agents dont le comportement est proche d'un humain ce qui est typique du réalisme. De l'autre, nous étudions le comportement d'un humain dans un jeu vidéo, qui n'a bien souvent rien à voir avec le comportement d'un humain dans le monde réel, ce qui se rapproche de la crédibilité. Par contre, notre mesure sera quasi inefficace pour mesurer le niveau d'intelligence d'un agent.

4.2 Protocole

Le protocole d'évaluation se compose de 4 points que nous détaillerons par la suite :

1. Définition des signatures et des distances entre signatures ;
2. Mesure des signatures pour un groupe de joueurs humains ;
3. Mesure des signatures pour un groupe d'agents ;
4. Calcul des distances entre les signatures des agents et des humains.

Il convient tout d'abord d'expliquer ce que nous entendons par signature. Une signature est un vecteur caractérisant en partie le comportement d'un personnage virtuel dans son environnement. La définition de celles-ci doit être menée avec rigueur pour garantir la qualité des résultats. Il faut en effet qu'elles aient un faible écart pour des comportements assez similaires. Cependant ces signatures doivent permettre de détecter des comportements clairement artificiels, il peut donc être intéressant d'avoir un exemple de ce genre de comportement lors de cette étape.

Pour déterminer le degré de ressemblance, nous allons utiliser les distances entre signatures, le choix du type de distance pour chaque signature est donc primordial. La définition des signatures et des mesures doit donc se faire simultanément. Pour définir de bonnes signatures il est parfois nécessaire de refaire plusieurs fois les étapes 1, 2 et 3 avec de petits jeux de tests. Il peut être intéressant, en outre, de chercher à avoir des distances entre une signature d'un agent et d'un humain qui aient un sens. En d'autres termes que les résultats obtenus puissent guider les chercheurs pour améliorer le modèle. Cependant, comme dans toute mesure de distance entre vecteurs, il faut prendre garde à la malédiction de la dimensionnalité [Köppen 00], les vecteurs doivent donc être de dimension raisonnable.

Pour déterminer les signatures des humains, il faut pour cela mesurer les comportements qu'ils adoptent pour le contrôle des personnages virtuels. Le panel d'humains doit être assez important pour être représentatif. La principale difficulté pour les jeux vidéo est qu'il existe une grande disparité de « compétences » : certains sont des joueurs assidus et d'autres non. Leurs comportements peut être très différents, il faut donc déterminer à l'avance à quel type de joueur l'agent doit ressembler. Il peut être intéressant de supprimer les *outliers*[11] qui

[11] *outlier* : valeur qui est beaucoup plus éloignée de la valeur moyenne que les autres.

peuvent être révélateurs de comportements étranges dans le jeu mais cela relève d'une étude au cas par cas.

Il faut ensuite déterminer les signatures pour les agents. Les conditions dans lesquelles les mesures sont faites doivent être au plus proches de celles de l'étape précédente. Il n'est évidemment pas possible de recréer les mêmes conditions, puisque les agents devront jouer contre d'autres agents et non contre des humains pour que l'expérience puisse être automatisable.

Enfin, l'évaluation en elle même, se fait en mesurant les distances entre les signatures des agents et celles des humains. Ici, de nombreuses possibilités s'offrent à nous. Nous pensons qu'utiliser une valeur moyenne n'est pas très approprié puisqu'une moyenne de signatures ne correspond vraisemblablement pas un comportement moyen. Il vaut mieux évaluer la distance entre chaque agent et chaque humain pour avoir des résultats plus significatifs. Plus la plus petite distance entre un agent et un humain est faible meilleur est le comportement de l'agent.

Les résultats obtenus lors de ce protocole ont deux intérêts principaux. Les signatures des agents peuvent permettre de déboguer ou de modifier le modèle si elles relèvent des anomalies ou des lacunes. L'évaluation quant à elle peut permettre d'améliorer le modèle et les paramètres de celui-ci. L'utilisation de l'approche que nous proposons peut donc servir tout au long du développement d'un modèle de comportement et de son codage pour améliorer la crédibilité finale de l'agent.

4.3 Surveillance des personnages virtuels

Pour pouvoir déterminer des signatures il faut pouvoir surveiller les personnages virtuels dans le jeu de manière automatique. Pour que cela soit possible à la fois avec un personnage contrôlé par un humain ou par un agent, il faut prendre un point de vue extérieur au personnage virtuel. Nous avons choisi d'« espionner » les perceptions et les actions (figure 1) d'un personnage afin de déterminer ses signatures. Cela permet d'avoir un point de vue extérieur tout en ayant accès à toutes les informations dont dispose le personnage virtuel.

Les signatures basées sur les perceptions permettent de mesurer la crédibilité des perceptions. Cependant un problème subsiste, puisque les perceptions que nous mesurons ne correspondent pas aux perceptions que l'agent ou l'humain interprète. Pour déterminer si ces perceptions sont interprétées il faut étudier les réactions du personnage et donc ses actions. Cela limite la quantité d'information que l'on peut en tirer car il faut deviner les décisions prises, n'y ayant pas accès. Par exemple, on peut déterminer si un ennemi a été vu ou non si le personnage lui tire dessus. Le problème est que ce n'est pas forcement une équivalence, si on le suppose, les mesures seront moins fiables.

En ce qui concerne les décisions, nous n'avons aucune mesure puisqu'elles sont prises soit par un programme extérieur, soit par un cerveau. Il est très difficile de déterminer des signatures sur quelque chose qu'on ne peut absolument pas mesurer. Il est cependant possible d'avoir des signatures qui lient perceptions

et actions mais elles seraient sans doute totalement incapable de modéliser des décisions complexes.

Les signatures basées sur les actions sont beaucoup plus faciles à obtenir. L'unique problème avec celles-ci est qu'elles peuvent bien souvent varier de manière importante entre des individus de différent niveau de compétence dans le jeu. Cela peut être géré en choisissant avec soin les joueurs qui participent à l'expérience comme nous l'avons évoqué plus tôt.

Les signatures basées sur les perceptions et surtout sur les actions semblent les plus prometteuses. Même si aux premiers abords on peut penser que seules les décisions jouent un rôle dans un comportement crédible, ce choix n'est pas une concession. Des perceptions crédibles sont une condition nécessaire à un comportement crédible et les actions crédibles sont nécessaires pour que les joueurs ne puissent pas distinguer un agent d'un humain.

Le principe de la construction d'une signature est simple : il suffit de mesurer différentes variables pendant l'expérience pour construire un vecteur représentatif de ces mesures. Par exemple on peut mesurer la position et la rotation d'un personnage virtuel, les collisions, etc. Ces mesures sont faites en temps réel mais rien n'empêche de traiter ces données une fois l'expérience terminée. Il est d'ailleurs généralement plus facile de faire l'évaluation sur des données normalisées, qui permettent de s'affranchir du nombre d'observation, ce qui nécessite d'avoir toutes les données.

5 Développement d'outils de surveillance des personnages virtuels d'Unreal Tournament 2004

Pour déterminer les signatures, il est nécessaire de pouvoir surveiller les perceptions et actions des personnages virtuels. Notre choix s'est naturellement tourné vers Pogamut 2 ayant de bonnes connaissances sur cet outil. De plus l'architecture est assez souple pour se prêter aux modifications que nous souhaitions faire et elle comporte des éléments qui peuvent être facilement réutilisés.

5.1 Modifications apportées à Pogamut 2

Pogamut 2 est prévu à l'origine pour permettre l'ajout d'agents dans Unreal Tournament 2004 pour pouvoir les contrôler grâce à un programme informatique. Notre objectif est différent puisque nous souhaitons pouvoir surveiller les perceptions et actions de tous les personnages virtuels dans le jeu. Cependant il y a des similarités entre l'objectif de Pogamut 2 et le notre, à savoir que Pogamut 2 permet l'exportation des perceptions du personnage virtuel contrôlé vers le programme de gestion des comportements. Nous avons donc exploité la partie exportation de données en oubliant la partie contrôle à distance. Restait à permettre la connexion de Pogamut 2 à n'importe quel personnage virtuel, même ceux contrôlés par des humains.

L'architecture globale de notre version modifiée est globalement identique à celle de l'originale (figure 7), l'intelligence artificielle étant remplacée par un programme de surveillance. Pour l'architecture du client (figure 8), là aussi il y a peu de changements, les principales modifications sont dans le langage de commandes : il n'est pas possible au programme de surveillance de demander l'exécution d'actions. Par contre il est toujours possible d'envoyer des commandes qui peuvent aider à la surveillance, comme demander le lancer de rayons pour analyser les perceptions vers une direction bien précise par exemple.

5.1.1 Modification de Gamebots

La première étape de la réalisation a été de modifier la partie Gamebots pour permettre d'exporter les perceptions de tous les personnages virtuels. Comme le montre le diagramme de classes simplifié (figure 14), l'environnement virtuel est représenté par la classe `LevelInfo`. Chaque « entité intelligente » est représentée par un objet de la classe `Controller`. Les personnages eux-mêmes sont représentés par la classe `Pawn`. L'environnement, personnages exclus, et les règles qui le régissent sont définis dans la classe `GameInfo`. Bien sur, comme dans tout langage objet, l'héritage permet de modifier le comportement des classes, nous ne détaillerons pas l'intérêt de chacune de ces modifications.

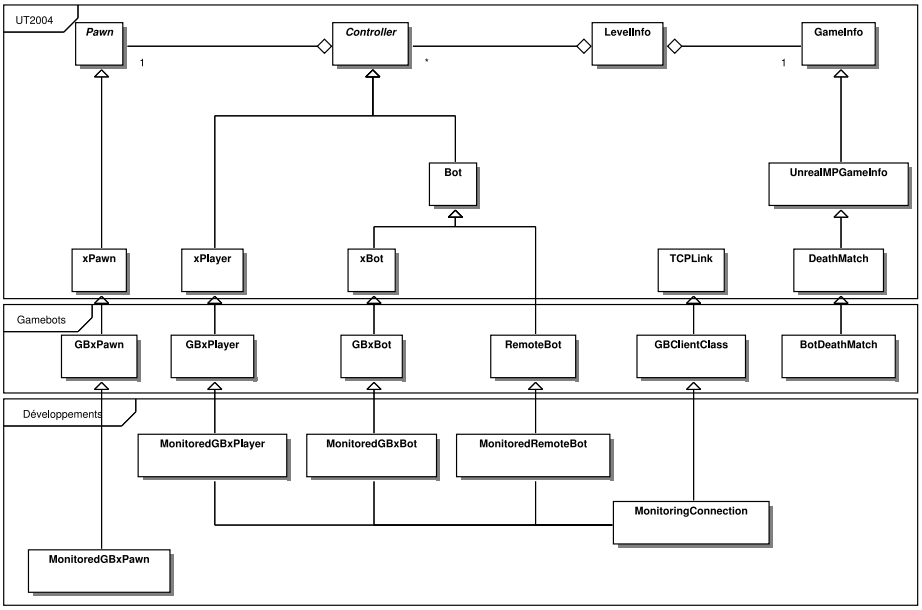


FIG. 14 – Diagramme de classes simplifié de Gamebots modifié pour surveiller les personnages virtuels dans Unreal Tournament 2004. Le cadre noté UT2004 regroupe certaines classes d'origine du jeu. Le cadre Gamebots regroupe certaines classes de Gamebots (dont certaines ont été modifiées lors des développements). Enfin le cadre développements regroupe les classes ajoutées lors de nos développements.

Le fonctionnement habituel et simplifié de Gamebots est le suivant : un port est ouvert sur le serveur de jeu qui permet la connexion d'une application extérieure (Pogamut 2). Lorsque ce programme se connecte, un objet de type `RemoteBot` et `GBxPawn` sont créés et sont ajoutés à l'environnement virtuel courant. Le contrôle du `GBxPawn`, le personnage virtuel, se fait par le programme distant via `RemoteBot`.

Dans notre cas, nous nous intéressons principalement aux méthodes UnrealScript correspondant aux perceptions et aux actions qui se trouvent à la fois dans la classe `Pawn` et `Controller`. Cependant, comme on aurait pu s'y attendre, les classes permettant la surveillance (`Monitored*`, classes dont le nom commence par Monitored) n'héritent pas directement de ces deux classes pour deux raisons :

- Les classes héritant directement de `Controller` sont définie *native* ce qui signifie qu'il faut avoir le code source C++ (payant) pour pouvoir les recompiler ;
- Les classes héritant de `Controller` et de `Pawn` n'appellent pas toujours les méthodes qu'elles redéfinissent de leur classe mère (via `super`). Il n'est donc pas possible d'« espionner » les appels de méthodes grâce à une classe mère, mais seulement grâce à une classe fille.

Les classes `Monitored*` fonctionnement toutes sur le même principe : les fonctions correspondant à une action ou une perception envoient un message et appellent la méthode parente. Bien entendu, les classes `Monitored*` héritant de `Controller` ont toutes un personnage virtuel de type `MonitoredGBxPawn`. Enfin, `MonitoringConnection` permet l'envoi des actions et perceptions capturées.

5.1.2 Modification de Pogamut 2

Le parser a dû être légèrement modifié pour introduire un nouveau message dans le protocole de Gamebots. Cela permet d'établir une nouvelle connexion de type surveillance qui n'existait pas auparavant. L'ajout a été très simple puisque Pogamut 2 utilise jFlex, un générateur d'analyseur lexical pour Java. Les autres messages ont été laissés à l'identique car nous utilisons les mêmes messages de perceptions et d'actions que ceux pour les agents Pogamut 2.

Le client a lui aussi été modifié pour répondre à nos besoins comme le montre la figure 15. La classe `MonitoredPlayer` regroupe les 4 structures de données comme précédemment à la différence qu'elle contient un `MonitoredPlayerBody` au lieu d'un `AgentBody`. `MonitoredPlayerBody` s'occupe de la mise à jour des structures de données comme sa classe mère, le changement étant dans l'initialisation car le protocole de Gamebots a été modifié.

5.2 La surveillance dans Pogamut 2

Pour mettre en place une évaluation il faut donc créer une classe qui hérite de la classe `MonitoredPlayer`. Son principe est identique à la classe `Agent` à savoir que la fonction `doLogic` est appelée plusieurs fois par seconde. Il faut donc redéfinir cette fonction pour calculer les signatures que l'on désire grâce aux structures de données qui sont mises à jour. Un point que nous avions omis

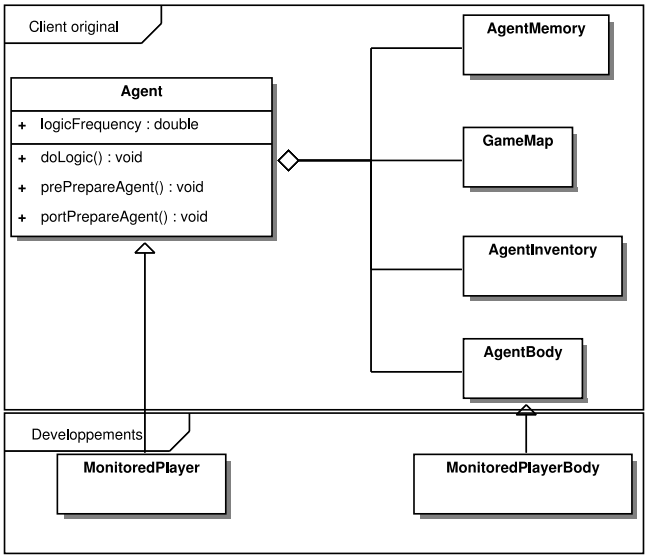


FIG. 15 – Diagramme de classe simplifié d'un agent surveillé dans Pogamut 2. Le cadre client original regroupe les classes déjà présentes dans Pogamut 2. Le cadre développements regroupe les développements que nous avons effectués.

de préciser plus tôt est qu'il est aussi possible d'exécuter des lignes de code lors de certains évènements. Par exemple, il est possible de mesurer certaines perceptions lorsque qu'il y a collision entre le personnage virtuel et un autre personnage par exemple.

En plus de la surveillance et du calcul des signatures il est nécessaire de coder la mesure des distances. Cela permet d'évaluer les agents par rapport aux humains. Comme nous l'avons précisé plus tôt, ce calcul peut se faire pendant l'expérience auquel cas il faut faire attention aux modifications sur les signatures étant mise à jour plusieurs fois par seconde.

Le plus simple est de faire l'évaluation après l'expérience afin de pouvoir traiter les données. Il est d'ailleurs nécessaire de sauvegarder les signatures obtenues afin de pouvoir les comparer à celles qui seront obtenues plus tard. Cette sauvegarde se fait très facilement en Java grâce à la sérialisation qui permet de générer un fichier contenant le ou les objets correspondant aux signatures. Il est alors très simple de pouvoir comparer les signatures des agents aux signatures sauvegardées des humains de référence.

Le projet Pogamut 2 modifié et prêt à l'emploi pour la surveillance des personnages d'Unreal Tournament 2004 est disponible à l'adresse `svn://artemis.ms.mff.cuni.cz/pogamut/branches/fabien_tence`. Son utilisation est soumise aux mêmes règles que le projet Pogamut 2, à savoir qu'il est possible de faire ce que vous voulez avec la plate-forme tant que vous citez Pogamut 2 et que vos travaux ne sont pas utilisés à des fins commerciales ou militaires.

6 Application : exemple d'évaluation

Pour valider notre méthode et pour montrer concrètement en quoi consiste l'évaluation, nous avons mené une évaluation test. Celle-ci n'est malheureusement pas complète par manque de temps mais elle permet de montrer quels sont les problèmes que l'on peut rencontrer et quels sont les points importants du protocole.

6.1 Protocole de l'expérience

6.1.1 Signatures

Le premier point est la définition des signatures. Nous en avons choisi 5, basées sur les actions pour des raisons de simplicité. Elles nous semblent caractériser correctement les différents aspects de l'évolution d'un personnage virtuel dans son environnement. Cependant nos expériences n'étant pas arrivées à leur terme, il est possible que certaines signatures n'aient en réalité pas les caractéristiques nécessaires pour être jugées utiles. Voici les 5 signatures utilisées :

- Approximation de la probabilité d'immobilisation : vecteur de dimension 1. La valeur de la seule composante de ce vecteur est le rapport entre le nombre de mesures où le personnage est immobile et le nombre de mesures totale.
- Approximation de la probabilité d'immobilisation sachant que le personnage est immobile : vecteur de dimension 1. La valeur de la seule composante de ce vecteur est le rapport entre le nombre de mesures où le personnage est immobile alors qu'il était immobile à la mesure précédente et le nombre de mesures où le personnage est immobile.
- Approximation de la distribution de probabilité de changement de direction : vecteur de dimension 20. Chaque composante $CdD_i, i \in \{0, ..., 19\}$ est égale au nombre de fois où on a mesuré un angle de i entre $\overrightarrow{D}_{t+1}$ et $\overrightarrow{D}_t$.
- Approximation de la distribution de probabilité de changement de direction du vecteur vitesse : vecteur de dimension 20. Chaque composante $CdV_i, i \in \{0, ..., 19\}$ est égale au nombre de fois où on a mesuré un angle de i entre $\overrightarrow{V}_{t+1}$ et $\overrightarrow{V}_t$.
- Approximation de la distribution de probabilité de la direction du vecteur vitesse par rapport à la direction : vecteur de dimension 20. Chaque composante $VrD_i, i \in \{0, ..., 19\}$ est égale au nombre de fois où on a mesuré un angle de i entre $\overrightarrow{V}_t$ et $\overrightarrow{D}_t$.

Les vecteurs $\overrightarrow{V}_t$, $\overrightarrow{V}_{t+1}$, $\overrightarrow{D}_t$ et $\overrightarrow{D}_{t+1}$ sont définis sur la figure 16. Les angles sont bien entendu définis sur une nouvelle échelle où 20 correspond à un tour complet. Cette échelle permet d'éviter la malédiction de la dimensionnalité si l'on avait pris des angles en degrés tout en gardant assez d'informations pour que les signatures soient utiles. Enfin, le terme de mesure correspond à un appel à la fonction `doLogic`.

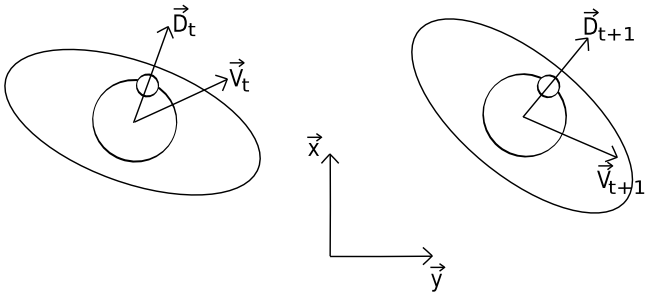


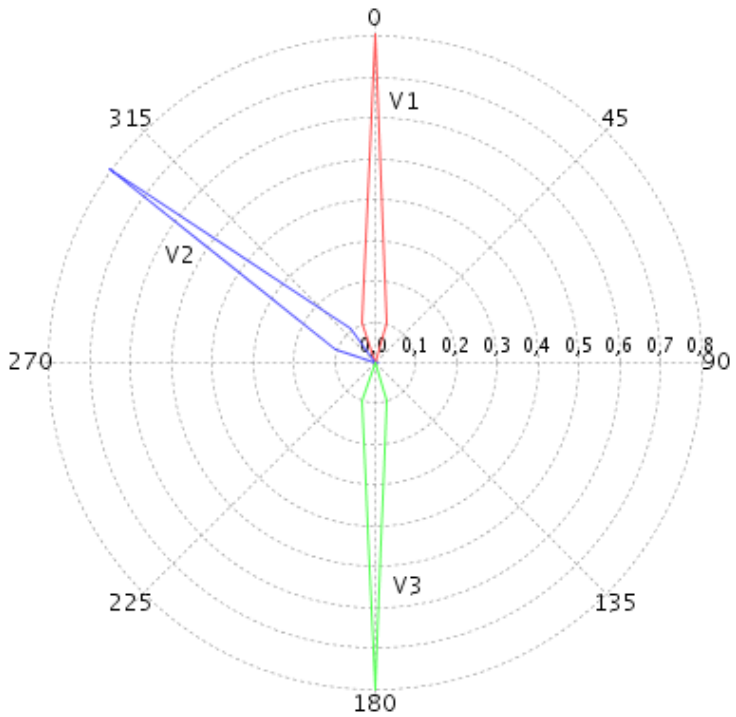
FIG. 16 – Vue du dessus d'un personnage virtuel à l'instant t et à l'instant $t+1$. $\overrightarrow{D}$ est le vecteur direction, il pointe toujours dans la direction où regarde le personnage. $\overrightarrow{V}$ est le vecteur vitesse, il n'est pas forcement colinéaire avec $\overrightarrow{D}$ puisqu'il est possible de faire des pas chassés dans Unreal Tournament 2004. Tous ces vecteurs sont projetés sur le plan $(\overrightarrow{x}, \overrightarrow{y})$ et normalisés pour plus de simplicité.

6.1.2 Distances entre signatures

Pour chacune de ces signatures, il faut ensuite définir une mesure de distance. Pour les vecteurs de dimension 1 nous avons préféré utiliser le carré de la différence plutôt que la différence. Cela permet de d'avoir une mesure très faible pour des valeur proches. Pour les vecteurs de dimension 20, nous avons choisi d'utiliser la *earth mover's distance* (EMD) [Rubner 00]. Cette mesure est souvent utilisée dans le domaine de l'image mais elle est bien adaptée dans notre cas. Son principe est simple : de manière imagée, sa valeur est l'effort minimum pour qu'un terrassier passe du « relief » d'un vecteur V_1 au « relief » d'un vecteur V_2. Nous l'avons adapté pour qu'elle se prête à des vecteurs de mesure d'angles, où la dernière composante est proche de la première. La figure 17 explique l'intérêt de cette mesure. Comme pour les signatures, les distances choisies ne sont peut-être pas les plus pertinentes cependant nous pensons évaluer leur intérêt leurs de prochaines expériences.

Pour déterminer le temps nécessaire pour une expérience, nous avons étudié l'évolution des mesures au cours du temps. Pour cela nous avons étudié les signatures de quatre agents participants à une partie de type *deathmatch*. La figure 18 montre un exemple de résultat que nous avons obtenu pour un de ces quatre agents. D'après ces résultats une expérience de 10 à 15 min permet d'avoir des mesures assez stables. Cette durée est tout à fait convenable puisque si des joueurs doivent participer à l'expérience, une partie de 15 min est une durée classique de partie.

Une autre caractéristique importante à vérifier est que deux comportements identiques aient des signatures proches. Nous avons donc comparé la distance entre les signatures de deux agents ayant les mêmes comportements en fonction du temps (figure 19). Cela permet en plus de déterminer de combien peuvent varier les signatures pour des comportements identiques, de vérifier si 10 à 15 minutes sont effectivement suffisantes pour arriver à une distance faible. Il res-



	EMD	L_2
(V_1,V_2)	3.0	env. 1.15
(V_1,V_3)	9.6	env. 1.15
(V_2,V_3)	7.0	env. 1.15

FIG. 17 – Exemple montrant l'intérêt de l'EMD par rapport à L_2. Les vecteurs n'ayant aucune composante composante commune, la distance L_2 est identique. Cependant, il est évident pour un humain que les vecteurs V_1 et V_2 sont les plus semblables. La distance EMD permet donc de mesurer cette similarité. Même si l'échelle du graphique va de 0 à 360 les vecteurs sont en réalité de dimension 20, comme ceux que nous utilisons pour les signatures.

sort de notre étude que la signature approximant la probabilité de la direction du vecteur vitesse par rapport à la direction est moins fiable que les autres signatures.

De cette même étude il ressort que la différence est encore un peu trop importante à 10 minutes et que 15 minutes sont préférables pour une expérience. Dans l'idéal, plus l'expérience est longue plus les résultats sont satisfaisants, mais cela n'est pas envisageable lorsque l'on fait participer des humains. Enfin un dernier point intéressant est que les variations observées dans la première étude ont une bien plus faible amplitude que les différences entre deux comportements similaires. C'est donc plutôt la diminution des distances entre signatures de comportements similaires qui est notre objectif.

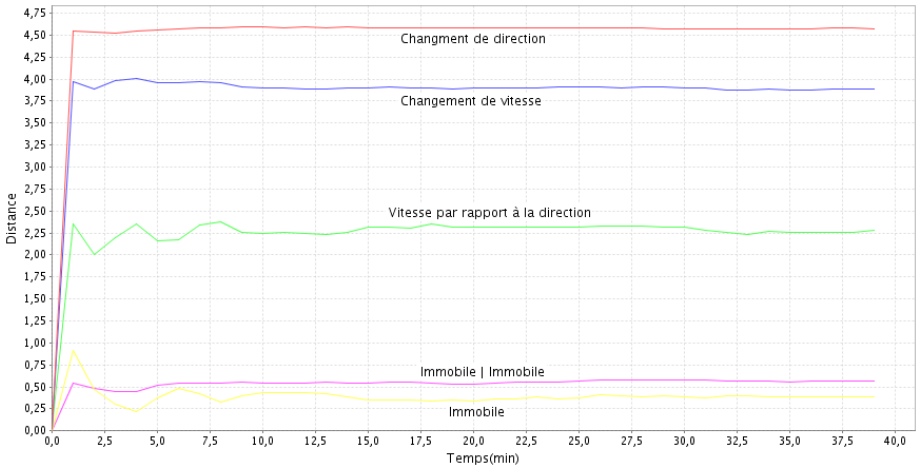
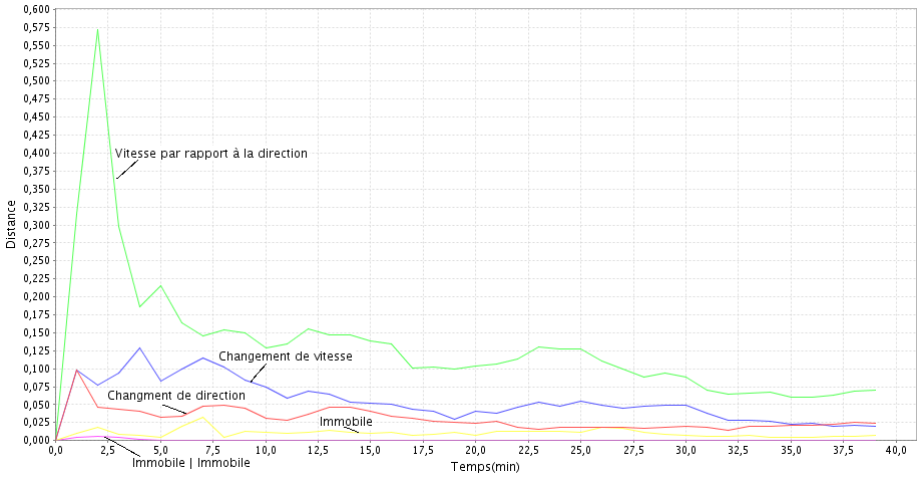


FIG. 18 – Évolution de la distance entre les signatures d'un agent et des signatures qui n'évoluent pas en fonction du temps. L'approximation de la probabilité d'immobilité à été multipliée par 100 pour pouvoir être comparée aux autre mesures. Les mesures se stabilisent clairement après 10-15 minutes de jeu.



directement sur les valeurs

FIG. 19 – Évolution de la distance entre les signatures de deux agents identiques en fonction du temps. Ici aussi, l'approximation de la probabilité d'immobilité à été multipliée par 100 pour pouvoir être comparée aux autre mesures. Plus le temps passe, meilleures sont les mesures, cependant on peut admettre qu'à partir de 15 minutes les mesures ont une différence assez faible pour être acceptables.

Il est à noter que les scores fournis par le jeu varient eux aussi pour des comportements identiques. Pour donner un ordre d'idée, dans une partie de type *deathmatch* ils peuvent varier d'environ 25% par rapport à la valeur moyenne après 1 heure de jeu alors que nos mesures varient de moins de 5% dans les

mêmes conditions (variations calculées sur les distances obtenues par rapport à des signatures de référence). Nos résultats sont donc plutôt encourageants.

De nombreuses autres études peuvent être menées sur les signatures et les distances. Par manque de temps nous n'avons réalisé que les 2 plus importantes, mais il serait intéressant d'étudier l'influence des facteurs suivants :

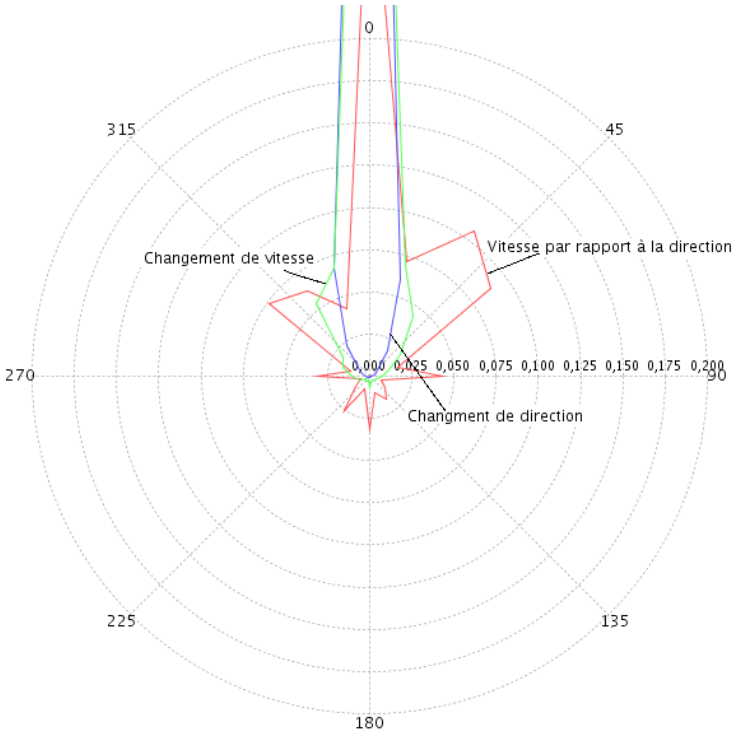
- Les règles du jeu, comme nous l'avons expliqué auparavant, Pogamut 2 est compatible avec 4 règles du jeu différentes. Certaines règles peuvent introduire la notion de rôle, chaque joueur d'une équipe pouvant avoir une fonction. Cela peut influencer de façon importante les signatures puisqu'un joueur qui s'occuperait de la défense aurait sans doute une signature d'immobilité importante ;
- L'environnement, les parties peuvent se dérouler dans des environnements différents. La géométrie ou topographie de ceux-ci pourraient avoir une influence sur les déplacements des joueurs et donc sur les signatures d'action ;
- Les caractéristiques des périphériques, les joueurs humains utilisent des périphériques d'entrée et de sortie pour interagir avec l'environnement virtuel. Ceux-ci ont une influence considérable sur les signatures comme nous le verrons plus loin ;
- L'expérience des joueurs humains dans les jeux vidéo et dans le jeu utilisé pour l'expérience. Ce facteur a sans doute l'influence la plus importante car la différence de comportement entre un joueur novice et un expert est très importante.

6.1.3 Détermination des signatures des joueurs humains

Pour le calcul des signatures des joueurs humains, nous avons fait l'expérience suivante : pendant 20 minutes, 6 joueurs humains s'affrontent dans le jeu Unreal Tournament 2004. Les règles sont les règles de bases (*deathmatch*) où le but est de tuer le maximum de fois ses adversaires. Le choix de règles simples permet de ne pas craindre l'apparition de rôles parmi les joueurs ni de variations trop importantes dans les techniques de jeu. La partie se déroule dans un unique environnement virtuel, l'étude de ce paramètre étant prévu pour la suite des travaux. Le nombre de joueurs humains étant suffisant, nous n'avons pas introduit d'agents dans la partie.

Pour la partie réelle (en opposition au virtuel) de l'expérience, chaque joueur disposait d'un ordinateur pour jouer. Tous ont utilisé le clavier et la souris pour contrôler le personnage et tous disposaient d'écrans assez similaires pour le rendu du monde virtuel. Un serveur était dédié à la simulation de l'environnement virtuel ainsi qu'aux mesures pour le calcul des signatures. Les joueurs ne formant pas d'équipes, aucune consigne n'a été donné sur l'interdiction ou non des communications entre joueurs.

Les signatures des joueurs humains nous apportent déjà quelques informations intéressantes (cf. figure 20). Lorsque l'on étudie la vitesse par rapport à la direction, on s'aperçoit que cette signature est très fortement influencée par la manière de contrôler le personnage virtuel. La configuration habituelle d'un



directement

sur les valeurs

Approx. $P(Immo_t)$	0.0769
Approx. $P(Immo_t \mid Immo_{t-1})$	0.7681

FIG. 20 – Exemple de signatures obtenues pour un humain. Le graphique est tronqué pour la valeur 0 pour plus de lisibilité.

joueur est d'avoir 4 touches, une pour avancer, une pour reculer, une pour les pas chassés à droite et une pour les pas chassés à gauche. Sur la signature on retrouve clairement ces 8 « piques », correspondant aux 4 directions et leurs composées.

6.1.4 Détermination des signatures des agents

L'expérience pour déterminer les signatures des agents s'est déroulée comme suit : 6 agents codés par les développeurs du jeu ont joué les uns contre les autres lors d'une partie suivant les règles du *deathmatch*. Ces agents ont joué dans le même environnement virtuel que les joueurs. Les agents ne se lassant pas du jeu, nous avons fait durer les parties 1 heure afin d'avoir des signatures très représentatives. Nous admettons que la différence de temps de jeu ne nuit pas à la qualité des résultats, même si nous n'en avons pas la preuve formelle.

Pour répartir la charge de travail, le serveur dédié et les agents sont sur un ordinateur alors que la surveillance se fait sur une autre machine. Comme pour les humains aucune consigne n'est donné quant à la communication, cependant cela n'a aucune influence car les agents n'ont pas la capacité de communiquer.

Dans le jeu, il est possible de choisir le niveau de compétence de l'agent. Nous avons choisi 6 agents de niveau différent pour étudier l'impact de ce paramètre. Dans la suite, nous n'étudierons que les résultats du meilleur, du moins bon et d'un agent moyen pour limiter le nombre de résultats présentés.

6.2 Résultats pour les agents d'Unreal Tournament 2004

Les distances entre les signatures des joueurs et des agents sont donnés en annexe A. À titre de comparaison, les score obtenus dans le jeu sont eux aussi fourni dans l'annexe pour mieux apprécier les similitudes et les différences entre les deux approches.

La première chose que l'on peut tirer de ces mesures est qu'elles ne permettent pas de mettre en lumière une différence claire entre les comportements humains et ceux des agents. Seule la signature d'immobilisation (tableau 4) permet de mettre en évidence que l'agent 3 a un comportement assez différent des humains mais aussi des autres agents. Ces résultats peuvent avoir plusieurs explications. Premièrement, les agents évalués sont de bons agents dont le comportement est assez crédible. Il serait intéressant de combiner nos expériences avec un questionnaire pour déterminer si ces agents sont réellement crédibles et si non, quels attitudes permettent de les discerner des humains. Une deuxième explication, est que les signatures et distances que nous avons mises en place ne sont pas assez fines pour détecter les différences entre les humains et les agents. Il faudrait évaluer des agents aux comportements clairement non crédibles pour confirmer ou non cette hypothèse. Enfin, une dernière explication, la plus vraisemblable, est que les points que nous avons évalués (les déplacements principalement) ne sont pas ceux sur lesquels les joueurs détectent le comportement artificiel des agents d'Unreal Tournament. En effet, un des comportements caractéristiques des agents est de passer exactement par le même endroit lors de leurs déplacements (cela est dû à la méthode utilisée pour le *pathfinding*). Or, aucune de nos signatures ne détecte un tel comportement.

Nos mesures ont cependant des caractéristiques intéressantes. On note qu'il existe une tendance générale selon laquelle plus les joueurs ont un bon score, plus leurs distances au plus mauvais des agents augmente et plus celle au meilleur agent diminue. Ces résultats confirmeraient l'hypothèse que les comportements des agents d'Unreal Tournament sont correctement modélisés. Même si on pourrait penser que nos mesures n'apportent pas plus d'informations que les scores, cela confirme que les signatures définies, même si elles sont simples, détectent réellement une similitude entre des comportements.

L'étude des résultats n'est pas encore complète puisqu'il serait intéressant de faire des études statistiques par exemple. Une analyse en composantes principales pourrait aussi permettre de mieux visualiser la proximité entre les différents comportements nous allons donc essayer de mettre ceci en place.

7 Conclusion

Dans ce rapport, nous avons cherché des solutions polyvalentes et flexibles pour l'intégration et l'évaluation de comportements d'agents dans des jeux vidéo. Dans cet objectif, nous avons présenté Pogamut 2, plate-forme de développement, d'intégration et d'évaluation de comportements d'agents pour le jeu vidéo Unreal Tournament 2004. Celle-ci nous semble la plus adaptée pour permettre la validation de travaux de recherche puisqu'elle permet le prototypage rapide et aisé de comportements. Pour des applications plus complexes d'autres solutions peuvent s'avérer être plus adaptées, cependant n'étant pas expert dans le domaine de l'intelligence artificielle, Pogamut 2 nous a semblé la plate-forme idéale pour continuer nos travaux.

Nous avons en effet proposé une nouvelle méthode d'évaluation de comportements crédibles et fourni des outils dans Pogamut 2 pour mettre en place ces évaluations. L'intérêt de notre proposition repose dans son approche radicalement différente des habituels questionnaires. Nous avons préféré des mesures objectives aux avis subjectifs d'humains. Nous nous sommes donc écarté de la notion de crédibilité tout en gardant comme objectif d'aider les chercheurs à modéliser de tels types de comportements.

Les résultats proposés ne sont pas encore significatifs, n'ayant pas eu le temps de faire une étude pousée de nos premiers résultats. Nous ne pouvons donc pas encore affirmer la validité ou non de notre approche. Même s'il s'avère dans la suite de nos travaux que le protocole que nous avons défini ne donne pas de bons résultats nous avons néanmoins mis en avant l'intérêt de mesures objectives pour évaluer des comportements habituellement jugés subjectivement.

Pour effectuer nos expériences nous avons dû développer des outils pour « espionner » les comportements des personnages virtuels dans Unreal Tournament 2004 et notamment des personnages virtuels contrôlés par des humains. Cette fonctionnalité n'était en effet pas présente dans Pogamut 2 qui est seulement prévu pour contrôler des personnages virtuels grâce à des intelligences artificielles.

Dans la suite du stage, nous comptons nous attacher à l'étude des signatures et des distances pour améliorer les résultats obtenus dans notre exemple. Il serait ensuite intéressant d'utiliser cet exemple d'évaluation avec un algorithme génétique afin d'étudier les comportements obtenus grâce à nos nouveaux critères de sélection. La possibilité d'évaluer de grand nombre de comportements est en effet l'un des principaux avantages de la méthode proposée. Enfin la dernière étape pour établir l'utilité ou non de notre proposition est de l'utiliser dans des travaux de développement de comportements crédibles. Cependant cela va s'avérer difficile à mettre en place de par le temps et les ressources nécessaires.

Quelques soit les conclusions sur l'utilité de nos travaux, les outils que nous avons développés ne sont en rien spécifiques à l'évaluation telle que nous l'avons définie. La possibilité de surveiller le comportement de joueurs humains pourrait être très utile pour des agents capables d'apprendre par imitation par exemple.

D'autres applications peuvent être envisagée comme l'étude du comportements des joueurs en fonction de certains paramètres. Nous avons déjà par ailleurs abordé ce sujet lors de l'étude des signatures obtenues par les joueurs humains.

Ce stage a été très enrichissant à plusieurs niveaux. Tout d'abord sur le plan théorique en ce qui concerne le domaine de l'intelligence artificielle et plus précisément les comportements crédibles. Ensuite d'un point de vue plus technique sur la simulation d'environnements virtuels et le réseau. Enfin sur le plan humain grâce aux différentes discussions avec les chercheurs et ingénieurs du CERV.

Références

[Adobbati 01] R. Adobbati, A.N. Marshall, A. Scholer, S. Tejada, G.A. Kaminka, S. Schaffer & C. Sollitto. *Gamebots : A 3D Virtual World Test-Bed For Multi-Agent Research*. Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS, 2001.

[AII 08] *Engenuity Technologies Inc. : AI-Implant.* `http://www.ai-implant.com`, 06 2008.

[Bratko 01] I. Bratko. Prolog Programming for Artificial Intelligence. Addison Wesley Publishing Company, 2001.

[Brown 04] C. Brown, P. Barnum, D. Costello, G. Ferguson, B. Hu & M. Van Wie. *Quake II as a Robotic And Multi-Agent Platform*. Robotics and Vision Technical Reports, 2004.

[Bryson 01] J.J. Bryson. *Intelligence By Design : Principles Of Modularity And Coordination For Engineering Complex Adaptive Agents*. PhD thesis, Massachusetts Institute of Technology, 2001.

[Burkert 07] O. Burkert, R. Kadlec, J. Gemrot, M. Bída, J. Havlíček, M. Dörfler & C. Brom. *Towards Fast Prototyping Of IVAs Behavior : Pogamut 2*. Intelligent Virtual Agents, 7th International Conference, IVA, pages 17–19, 2007.

[Cavazza 03] Marc Cavazza, Fred Charles & Steven J. Mead. *Interactive Storytelling : From AI Experiment To New Media*. In ICEC '03 : Proceedings of the second international conference on Entertainment computing, pages 1–8, Pittsburgh, PA, USA, 2003. Carnegie Mellon University.

[FEA 08] *FEAR : Foundations for Genuine Game AI.* `http://fear.sourceforge.net/`, 06 2008.

[Fle 08] *About FlexBot.* `www.cs.northwestern.edu/~gdunham/flexbot/about.htm`, 06 2008.

[Heeter 92] Carrie Heeter. *Being There : The Subjective Experience Of Presence*. Presence : Teleoper. Virtual Environ., vol. 1, no. 2, pages 262–271, 1992.

[Jav 08] *JavaBot for Unreal Tournament.* `utbot.sourceforge.net/`, 06 2008.

[Kaminka 02] Gal A. Kaminka, Manuela M. Veloso, Steve Schaffer, Chris Sollitto, Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer & Sheila Tejada. *GameBots : A Flexible Test Bed For Multiagent Team Research*. Communications of the ACM, vol. 45, no. 1, pages 43–45, January 2002.

[Köppen 00] M. Köppen. *The Curse Of Dimensionality*. 5th Online World Conference on Soft Computing in Industrial Applications (WSC5), 2000.

[Laird 00] John E. Laird & Michael van Lent. *Human-level AI's Killer Application : Interactive Computer Games*. In AAAI Fall Symposium Technical Report, pages 80–97, 2000.

[Laird 01] John E. Laird. *It Knows What You're Going to Do : Adding Anticipation to a Quakebot*. In AGENTS '01 : Proceedings of the fifth international conference on Autonomous agents, pages 385–392, New York, NY, USA, 2001. ACM.

[Laird 02a] J.E. Laird. *Research In Human-Level AI Using Computer Games*. Communications of the ACM, vol. 45, no. 1, pages 32–35, 2002.

[Laird 02b] J.E. Laird, M. Assanie, B. Bachelor, N. Benninghoff, S. Enam, B. Jones, A. Kerfoot, C. Lauver, B. Magerko, J. Sheiman*et al.* *A Test Bed For Developing Intelligent Synthetic Characters*. Ann Arbor, vol. 1001, pages 48109–2110, 2002.

[Le Hy 07] Ronan Le Hy. *Programmation et apprentissage bayésien de comportements pour des personnages synthétiques, application aux personnages de jeux vidéos*. PhD thesis, Institut national polytechnique de Grenoble, 2007.

[Lepouras 04] G. Lepouras & C. Vassilakis. *Virtual Museums For All : Employing Game Technology For Edutainment*. Virtual Reality, vol. 8, no. 2, pages 96–106, 2004.

[Lester 97] J.C. Lester & B.A. Stone. *Increasing Believability In Animated Pedagogical Agents*. International Conference on Autonomous Agents : Proceedings of the first international conference on Autonomous agents, vol. 5, no. 08, pages 16–21, 1997.

[Loyall 97] A.B. Loyall. *Believable Agents : Building Interactive Personalities*. PhD thesis, Stanford University, 1997.

[Mac Namee 04] Brian Mac Namee. *Proactive Persistent Agents : Using Situational Intelligence to Create Support characters in Character-Centric Computer Games*. PhD thesis, Trinity College Dublin, 2004.

[Molineaux 05] M. Molineaux & D.W. Aha. *TIELT : A Testbed For Gaming Environments*. Proceedings of the Twentieth National Conference on Artificial Intelligence (Intelligent Systems Demonstrations), 2005.

[Qua 08] *The QuakeBot Page*. `www.unconventional-wisdom.org/QuakeBot/quakebot.htm`, 06 2008.

[Robert 05] Gabriel Robert & Agnès Guillot. *MHiCS, une architecture de sélection de l'action motivationnelle et hiérarchique à systèmes*

de classeurs pour personnages non joueurs adaptatifs. PhD thesis, Université Pierre et Marie Curie, 2005.

[Robillard 03] G. Robillard, S. Bouchard, T. Fournier & P. Renaud. *Anxiety and Presence during VR Immersion : A Comparative Study of the Reactions of Phobic and Non-phobic Participants in Therapeutic Virtual Environments Derived from Computer Games.* CyberPsychology & Behavior, vol. 6, no. 5, pages 467–476, 2003.

[Rubner 00] Y. Rubner, C. Tomasi & L.J. Guibas. *The Earth Mover's Distance As A Metric For Image Retrieval.* International Journal of Computer Vision, vol. 40, no. 2, pages 99–121, 2000.

[Schneider 03] Nicolas Schneider. Navigation hybride appliquée aux personnages non joueurs du jeu vidéo counter strike. Laboratoire d'Informatique de Paris 6, 2003.

[Silverman 06] B.G. Silverman, G. Bharathy, K. O'Brien & J. Cornwell. *Human Behavior Models For Agents In Simulators And Games : Part II : Gamebot Engineering with PMFserv.* Presence : Teleoperators & Virtual Environments, vol. 15, no. 2, pages 163–185, 2006.

[Slater 04] M. Slater. *How Colorful Was Your Day ? Why Questionnaires Cannot Assess Presence In Virtual Environments.* Presence : Teleoperators & Virtual Environments, vol. 13, no. 4, pages 484–493, 2004.

[Smith 00] S.P. Smith & D.J. Duke. *Binding Virtual Environments To Toolkit Capabilities.* Computer Graphics Forum, vol. 19, no. 3, pages 81–89, 2000.

[Smith 01] S.P. Smith & M.D. Harrison. *Editorial : User Centred Design And Implementation Of Virtual Environments.* Int. J. Human-Computer Studies, vol. 55, pages 109–114, 2001.

[Trenholme 08] D. Trenholme & S.P. Smith. *Computer Game Engines For Developing First-Person Virtual Environments.* Virtual Reality, 2008.

[Wilson 91] Stewart W. Wilson. *The Animat Path To AI.* In Jean-Arcady Meyer & Stewart W. Wilson, editeurs, From Animals to Animats, pages 15–21, 1991.

[XAI 08] *X-AIment GmbH : X-AItment.* `http://www.x-aitment.net/`, 06 2008.

A Résultats obtenus lors de l'expérience test

Les lettres A et J désignent respectivement Agent et Joueurs. Les indices de chaque joueur et agent indiquent leur niveau par rapport au critère de score défini dans le jeu : plus l'indice est petit meilleur est le joueur (cf. tableau 6).

	J 2	J 3	J 4	J 5	J 6	A 1	A 2	A 3
Joueur 1	0.085	0.074	0.073	0.031	0.074	0.051	0.105	0.077
Joueur 2		0.034	0.056	0.091	0.159	0.069	0.070	0.122
Joueur 3			0.046	0.079	0.146	0.047	0.066	0.108
Joueur 4				0.069	0.133	0.053	0.059	0.098
Joueur 5					0.070	0.042	0.105	0.077
Joueur 6						0.105	0.156	0.079
Agent 1							0.076	0.063
Agent 2								0.098

TAB. 1 – Distances entre des agents et joueurs pour la signature « approximation de la distribution du changement de direction ».

	J 2	J 3	J 4	J 5	J 6	A 1	A 2	A 3
Joueur 1	0.042	0.056	0.071	0.228	0.235	0.389	0.216	0.443
Joueur 2		0.056	0.062	0.219	0.226	0.416	0.229	0.435
Joueur 3			0.098	0.261	0.271	0.394	0.245	0.477
Joueur 4				0.165	0.175	0.435	0.248	0.381
Joueur 5					0.028	0.530	0.329	0.230
Joueur 6						0.537	0.339	0.212
Agent 1							0.211	0.745
Agent 2								0.535

TAB. 2 – Distances entre des agents et joueurs pour la signature « approximation de la distribution du changement de direction du vecteur vitesse ».

	J 2	J 3	J 4	J 5	J 6	A 1	A 2	A 3
Joueur 1	0.453	0.373	0.307	0.481	0.517	1.583	0.458	1.156
Joueur 2		0.318	0.260	0.933	0.942	1.197	0.202	1.512
Joueur 3			0.370	0.798	0.740	1.515	0.368	1.194
Joueur 4				0.703	0.760	1.345	0.273	1.399
Joueur 5					0.160	2.036	0.932	0.747
Joueur 6						2.097	0.963	0.682
Agent 1							1.162	2.710
Agent 2								1.548

TAB. 3 – Distances entre des agents et joueurs pour la signature « approximation de la distribution de la direction du vecteur vitesse par rapport à la direction ».

	J 2	J 3	J 4	J 5	J 6	A 1	A 2	A 3
Joueur 1	0.262	0.224	0.091	0.013	0.004	0.080	0.520	5.844
Joueur 2		0.971	0.662	0.159	0.329	0.632	0.044	3.632
Joueur 3			0.029	0.343	0.170	0.036	1.427	8.358
Joueur 4				0.172	0.058	0.000	1.047	7.396
Joueur 5					0.030	0.157	0.370	5.313
Joueur 6						0.049	0.613	6.147
Agent 1							1.009	7.295
Agent 2								2.878

TAB. 4 – Distances entre des agents et joueurs pour la signature « approximation de la probabilité d'immobilisation ». Les valeurs obtenues ont été multipliées par 100 pour plus de lisibilité.

	J 2	J 3	J 4	J 5	J 6	A 1	A 2	A 3
Joueur 1	0.938	0.013	0.110	0.226	0.228	0.039	0.242	1.362
Joueur 2		1.176	0.406	0.243	0.241	0.596	0.227	0.040
Joueur 3			0.200	0.350	0.352	0.098	0.370	1.646
Joueur 4				0.021	0.021	0.018	0.026	0.699
Joueur 5					0.000	0.078	0.000	0.478
Joueur 6						0.079	0.000	0.476
Agent 1							0.087	0.942
Agent 2								0.456

TAB. 5 – Distances entre des agents et joueurs pour la signature « approximation de la probabilité d'immobilisation sachant que le personnage est immobile ». Les valeurs obtenues ont été multipliées par 100 pour plus de lisibilité.

	Score
Joueur 1	75
Joueur 2	55
Joueur 3	48
Joueur 4	44
Joueur 5	38
Joueur 6	35

	Score
Agent 1	114
Agent 2	55
Agent 3	22

TAB. 6 – Scores obtenus dans le jeu. Agents et joueurs sont séparés car ils n'ont pas obtenus ces scores dans la même partie, les agents n'ayant jamais affronté les joueurs.